Comprando esta edição tenha acesso na WEB a:
+210 vídeo-aulas e +140 artigos

www.clubedelphi.net **Feita para desenvolvedores**

ClubeDelphi

Ano 6 · Edição 79 · R$8,90

DevMedia GROUP

ISSN 1517990-7

Controle de qualidade
QA Audits - faça um ckeck-up da qualidade do seu código-fonte

Mão na Massa – Aplicativo de recepção com WebCam
Construa uma aplicação que captura uma foto através da WebCam e exibe no formulário/armazena no banco de dados

VoIP com Delphi

Construa seu próprio 'Skype", que permita aos usuários:

- Conversar em tempo real através do micro (de graça)
- Realizar um 'videocast' com voz e vídeo da Webcam

Especial

Agendamento de backups no IB/FB

- FIBS: deixe seu backup automático e programado para acontecer diariamente
- Crie sua própria ferramenta de agendamento de backups

Componentes
Programe a validação de formulários de forma visual com a JVCL

Introdução ao ADO.NET
Conheça todas as formas de acesso a dados no ADO.NET

DataSnap
Configurando o Windows 2003 Server para aplicações DataSnap

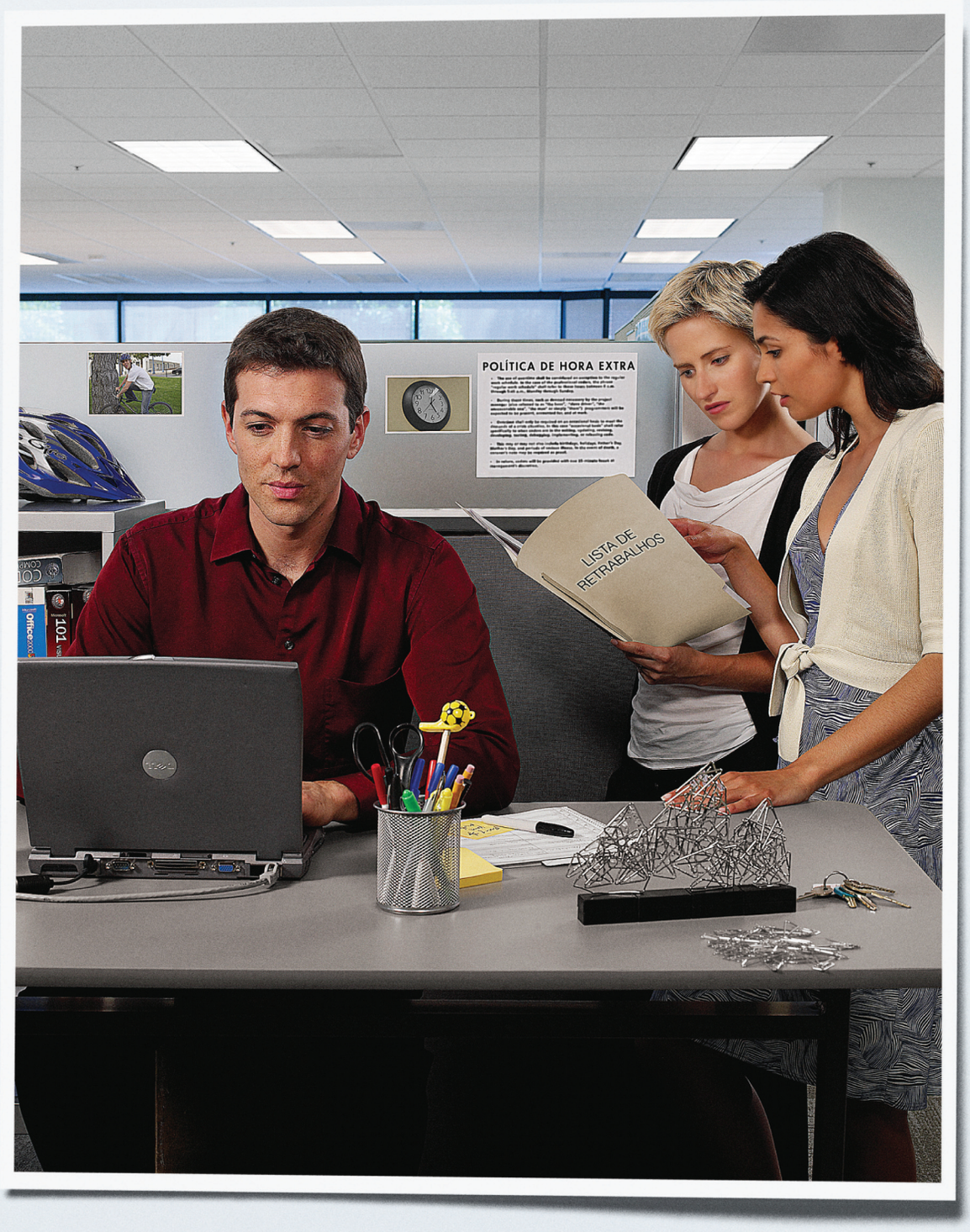
Coluna: Ask the Expert!
Afinal, o que é e para que serve um PChar?

POO
Automatize a serialização de objetos através do framework nTierObjects

Seu potencial. Nossa inspiração.™

Microsoft®

© 2006 Microsoft Corporation. Todos os direitos reservados. Microsoft, Visual Basic, Visual Studio, o logo Visual Studio e "Seu potencial. Nossa Inspiração." são marcas comerciais, registradas ou não, da Microsoft Corporation nos Estados Unidos e/ou outros países. Os nomes de companhias e produtos reais mencionados aqui podem ser marcas comerciais de seus respectivos proprietários.



Visual Studio® 2005.
A diferença é óbvia.

Viu a diferença? Os seus colegas também vão ver. Os atalhos mais rápidos do Visual Basic® 2005 permitem que você explore ao máximo as suas habilidades, ao mesmo tempo em que ajudam a solucionar os projetos mais desafiadores – aqueles que constroem a sua reputação pessoal. São mais de 400 features que facilitam a codificação, para você concentrar o seu trabalho no que realmente interessa.
Veja todas as 400 diferenças em **www.msdn.com.br/visualstudio2005**

Microsoft®
Visual Studio® 2005

NÃO PERCA

PORTAL DO ASSINANTE

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:

- Mais de 220 Vídeo Aulas!
- Diversos Mini-Cursos Gratuitos!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse www.devmedia.com.br/clubedelphi/portal.asp e utilize as informações abaixo:

Login: DVM.PL

Senha: A55LY

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Designer
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
jaxs_design@yahoo.com.br

Articulistas desta edição
Rodrigo Sendin, Marcos Alexandre Miguel, Marco Antônio Pereira Araújo, Fabricio Desbessel, Adriano Santos, Murilo Costa Monteiro Filho, Jorge Luis Bublitz, Rafaela de Campos, Rodrigo Otto Mostaert, Danilo Rocha Valente e Fernando Nomellini

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão
Fabio Correa
fabiocorrea@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:
Kaline Dolabella
publicidade@devmedia.com.br

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Thiago Andrade – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Parcerias

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:
Jeff Wendell
jeff@devmedia.com.br

ASSESPRO PARANÁ

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão! Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

ClubeDelphi

Ano 6 - 79ª Edição - 2006 - ISSN 1517990-7 **Impresso no Brasil**

Editorial

Acredito que o termo VoIP não seja novidade para mais ninguém. VoIP é uma tecnologia que permite a digitalização, codificação da voz e o encapsulamento desse dado em pacotes IP, para transmissão em uma rede que utilize o protocolo TCP/IP. Através dela, aplicações podem por exemplo codificar sua voz em forma de bits e trafegar pela Internet. Softwares como o Skype, MSN Messenger e Google Talk são bons exemplos de uso de VoIP. Permitem que pessoas conversem pela internet, usando seu próprio computador como se fosse uma aparelho telefônico, e o melhor de tudo, na maioria das vezes sem pagar um centavo. E que tal implementar seu próprio software de VoIP usando Delphi? É o que mostra o Jorge, em um artigo exclusivo para esta edição. Uma curiosidade: o próprio Skype (www.skype.com) foi feito em Delphi e é hoje uma das principais aplicações na área.

Ainda nesta edição, confira duas matérias sobre backups no IB/FB. No artigo da Rafaela, conheça a ferramenta FIBS (Firebird-Interbase Backup Scheduler), aplicativo Open Source voltado para backup de bancos Firebird e InterBase. Permite que o desenvolvedor ou DBA agende os backups a serem realizados, defina local de destino, mirrors, frequência (diariamente, semanalmente etc.). Indispensável para quem trabalha como IB/FB. Em outro artigo, o Rodrigo mostra como você pode construir sua própria ferramenta de backup utilizando os componentes nativos do Delphi, como a classe TIBBackupService. Ideal se você precisa embutir essa funcionalidade em uma aplicação existente.

O Rodrigo Sendin explora todos os detalhes relativos ao acesso a banco de dados no Delphi for .NET, através do ADO.NET. Como engine de acesso, usa o provider do próprio Firebird para ADO.NET. Conheça o que são e para que servem os Managed Providers, componentes de conexão, FbDataReaders, FbCommands, FbDataAdapters e DataSets.

E ainda nesta edição: DataSnap, validações, serialização de objetos e muito mais!

Um abraço a todos, boa leitura e muito sucesso com o Delphi.

Guinther Pauli – guinther@devmedia.com.br
MCP, MCAD, MCSD.NET, Delphi Certified

ÍNDICE

08 - Explorando as classes de acesso a dados no ADO.NET
Rodrigo Sendin

14 - QA Audits - faça um ckeck-up no seu código-fonte
Marcos Alexandre Miguel e Marco Antônio Pereira Araújo

22 - Validação simplificada com os controles da JVCL
Fabricio Desbessel

27 - Usando Webcam com Delphi
Adriano Santos

32 - Configurando o Windows 2003 Server para aplicações DataSnap
Murilo Costa Monteiro Filho

36 - VoIP com Delphi - Desenvolva o seu próprio "Skype"
Jorge Luis Bublitz

40 - Automatizando backup no Firebird
Rafaela de Campos

44 - Criando uma ferramenta personalizada para backup no IB/FB
Rodrigo Otto Mostaert

48 - Serialização de objetos
Danilo Rocha Valente e Fernando Nomellini

Ask the Expert

Perguntas e Respostas

PChar?

O que é PChar? Vejo muito esse tipo de campo nos códigos Delphi, mas não sei o que posso fazer com ele.
Juliano Caldas

PChar nada mais é que um ponteiro para um *Array de Char*. O seu último caractere é um *Null*, *Chr(0)*, por isso também é chamado de *null-terminated string*. A primeira vista pode parecer simplesmente uma *string* que termina com *Chr(0)*, mas ao contrário da *string*, com *PChar* você deve tomar cuidado com alocação de memória, por ser um ponteiro.

Usamos esse tipo "esotérico" principalmente em duas situações: chamadas à *APIs* do Windows e chamadas a funções que estão em DLLs escritas em *C/C++*. Se a função pede um parâmetro do tipo *PChar* mas apenas lê seu conteúdo, você pode usar *strings* normalmente. Veja o exemplo a seguir, usando a função *MessageBox* do Windows:

```
procedure SayHello;
var
  sNormalString: string;
begin
  sNormalString:= 'Hello!!';
  MessageBox(0, PChar(sNormalString),
    'Constante String', 0);
end;
```

Se a função pede um *PChar* como parâmetro e altera o seu conteúdo, você deverá alocar memória. Após o retorno da função é muito fácil converter um *PChar* em *string*. Em *Delphi*, um *Array de Char* é compatível com *PChar*, com a vantagem de já alocar memória. Veja o exemplo a seguir, que retorna o caminho da pasta temp do Windows:

```
function MyGetTempPath: string;
var
  caminho: array[0..MAX_PATH] of Char;
  { MAX_PATH é uma constante do Windows }
begin
  GetTempPath(MAX_PATH, caminho);
  Result := caminho;
end;
```

Existem casos em que você não sabe o tamanho da memória a ser alocada. Nesses casos deve-se alocá-la dinamicamente. Veja o exemplo a seguir, que retorna o nome do usuário logado no Windows:

```
function MyGetUserName: string;
var
  nome: PChar;
  tam: DWORD;
begin
  nome := nil;
  { Pega o tamanho necessário para o Buffer }
  tam := 0;
  GetUserName(nome, tam);
  { Aloca memória (o #0 final é incluído em "tam") }
  nome := StrAlloc(tam);
  try
    { Chama a função novamente }
    GetUserName(nome, tam);
    Result := nome;
  finally { Limpa o buffer }
    StrDispose(nome);
  end;
end;
```

Resposta enviada por **Jorge Bublitz** (*bublitz@hotmail.com*).

Portal ClubeDelphi PLUS

+200 vídeo aulas

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 200 vídeo aulas; ii) diversos mini cursos gratuitos; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 140 artigos exclusivos (que não foram publicados na revista)!;

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

Artigos

SQL Server - Criando banco, tabelas e acesso no Delphi - Parte I e II

Veja os artigos de Luciano Pimenta, que mostram como criar banco, tabelas e acesso ao SQL Server 2000 com o Delphi.

Curso de dbExpress e DataSnap - Parte XXIV a XXX

Veja os artigos finais do curso de Guinther Pauli, sobre dbExpress e DataSnap.

Vídeo-Aulas

Trabalhando com o IBExpert

Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com a ferramenta de manutenção de bancos IB/FB.

Imprimindo os registros selecionados em um DBGrid com o QuickReport

Veja nessa vídeo aula de Luciano Pimenta, como imprimir registros selecionados em um DBGrid com o Quick Report.

Código de barras no Rave Reports

Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com código de barras no Rave.

Minicurso SysCash - Parte IV

Veja nessa vídeo aula de Everson Volaco, mas um capítulo que mostra o sistema SysCash.

Imprimindo os registros selecionados no DBGrid com Rave

Veja nessa vídeo aula de Luciano Pimenta, como imprimir somente os itens selecionados em um DBGrid com o Rave.

Código de barras no Quick Report

Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com código de barras no Quick Report.

Global Pages no Rave Reports

Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com páginas globais no Rave.

Alterando relatórios no Rave em tempo de execução

Veja nessa vídeo aula de Luciano Pimenta, como dar a opção ao usuário alterar relatórios no Rave.

Trabalhando com Web Services no Delphi 2006

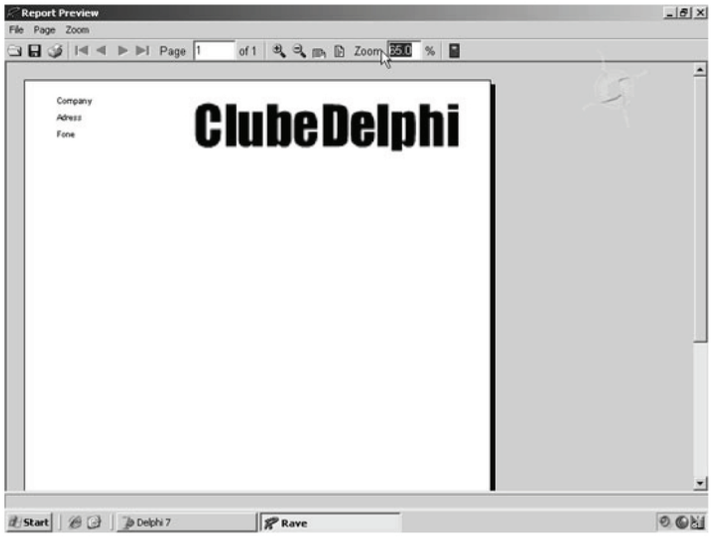
Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com Web Services no Delphi 2006.

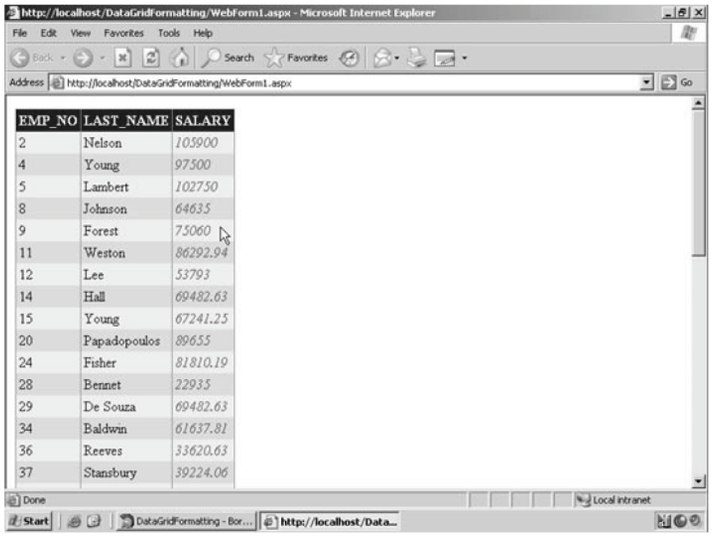
Mostrando imagens no DataGrid

Veja nessa vídeo aula de Luciano Pimenta, como mostrar imagens em um DataGrid do ASP.NET.

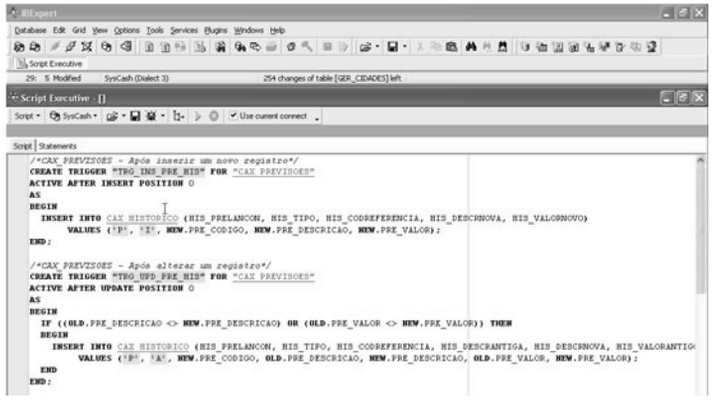
Preenchendo o DataGrid com informações de diretórios

Veja nessa vídeo aula de Luciano Pimenta, como listar arquivos e diretórios no DataGrid.





EMP_NO	LAST_NAME	SALARY
2	Nelson	105900
4	Young	97500
5	Lambert	102750
8	Johnson	64635
9	Forest	75060
11	Weston	86292.94
12	Lee	53793
14	Hall	69482.63
15	Young	67241.25
20	Papadopoulos	89655
24	Fisher	81810.19
28	Bennet	22935
29	De Souza	69482.63
34	Baldwin	61637.81
36	Reeves	33620.63
37	Stansbury	39224.06



+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos

TOTALMENTE GRÁTIS

www.delphi.eti.br

Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

Explorando as classes de acesso a dados no ADO.NET

RODRIGO SENDIN
(rodrigo.sendin@terra.com.br)
é tecnólogo formado pela FATEC-AM. Há 10 anos atua com desenvolvimento de software, e atualmente trabalha na TauNet Consulting como desenvolvedor C#, em projetos de Workflow, SharePoint, ASP.NET, Business Inteligence, e Knowledge Management.

Neste artigo veremos a biblioteca do .NET responsável pelo acesso à banco de dados, o ADO.NET. O ADO.NET pode ser utilizado por qualquer linguagem compatível com o .NET, inclusive o Delphi. O ADO.NET é um modelo de objetos composto por diversas classes, que podem ser divididas em dois grandes grupos: *Data Classes,* que são capazes de conter dados recuperados de bancos, e os *Managed Providers,* que fazem o acesso aos dados. Com o primeiro grupo, podemos trabalhar de forma desconectada do banco, e com o segundo trabalhamos de forma conectada.

Managed Providers

Vamos começar com os *Managed Providers*, que são utilizados para acessar os dados diretamente no banco. Os *Managed Providers* do ADO.NET são divididos em famílias. Cada família é responsável por acessar um tipo de fonte de dados, são elas: SQL Server, Firebird, OLEDB, Oracle, ODBC, BDP etc. Nos exemplos deste artigo utilizaremos as classes da família Firebird, mas os exemplos são os mesmos, caso queira utilizar os outros *providers* basta utilizar o respectivo *namespace*.

FbConnection

Vamos começar nosso exemplo criando um novo projeto ASP.NET no Delphi for .NET. Inclua um *Button* e um *Label* na página *WebForm.aspx*. Esse *Button* deverá chamar-se "btn-

Instalando o Firebird .NET Provider

Baixe o arquivo do endereço *www.firebirdsql.org/index.php?op=files&id=netprovider* (para a versão 1.1 do .NET). Após a instalação, para adicionar os componentes no Delphi, acesse o menu *Component>Installed Components*, clique no botão *Select an Assembly* e escolha o arquivo *FirebirdSql.Data.Firebird.dll* no diretório *C:\Arquivos de programas\FirebirdNETProvider[versão].*

Connection" e sua propriedade *Text* deverá ser "Connection".

Adicione um *FbConnection* no *Web Form* e faça a configuração para o banco de dados *Employee.fdb* que acompanha a instalação do Firebird. O *FbConnection*, logicamente, é utilizado para estabelecer uma conexão com o banco.

Insira no evento *Click* do nosso *Button,* o código da **Listagem 1**.

Listagem 1. Exemplo de utilização do FbConnection

```
try try
  FbConnection1.Open;
  Label1.Text := Label1.Text +
    '<li>Conexão Estabelecida com Sucesso!';
  except
      Label1.Text := Label1.Text +
      '<li>Falha na Conexão!';
  end;
finally
  FbConnection1.Close;
  Label1.Text := Label1.Text +
    '<li>Conexão Fechada!';

end;
```

No código da listagem anterior, chamamos o método *Open*, utilizado para abrir a conexão com o banco. Caso a conexão não seja estabelecida por algum problema na *string* de conexão ou um problema com o próprio banco, teremos uma ocorrência de uma exceção no *try..except*.

E dentro do *try..finally* estamos chamando o *Close*, responsável por fechar a conexão com o banco. Um dos grandes problemas que encontramos nas aplicações Web são as conexões que esquecemos abertas, é para isso que devemos utilizar uma estrutura *try..finally*, e chamarmos o método *Close* dentro do *finally*. Assim, mesmo que ocorra uma exceção, o *Close* será executado e a conexão será fechada (ou devolvida para o pool de conexões, caso esteja usando essa opção).

Faça um teste executando o projeto e clicando no botão *Connection*. Veja na **Figura 1** que as mensagens no *Label* indicam o código que foi executado.

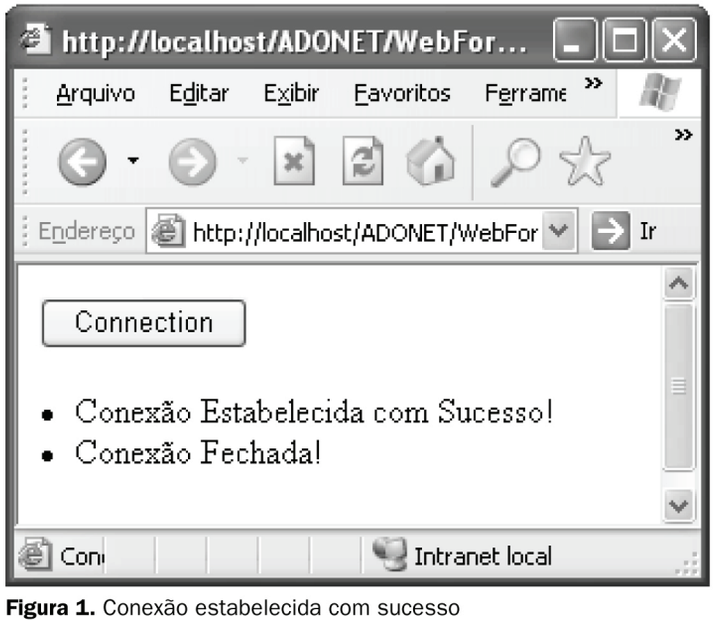


Figura 1. Conexão estabelecida com sucesso

Como não ocorreu nenhuma exceção, o código do bloco *try.. except* não foi executado. Feche o browser e faça um outro teste, substituindo a *string* de conexão no editor, por uma inválida. Salve, compile e execute novamente o projeto. Veja que nesse caso (**Figura 2**), o código do bloco foi executado e a linha logo após o método *Open* não foi executada devido à exceção ocorrida.

ExecuteScalar

A classe *FbCommand* é utilizada para disparar comandos SQL no banco e possui três métodos principais para realizar essa tarefa. Vamos começar analisando o *ExecuteScalar*. Inclua um novo botão ao lado do *Connection*, chamado "btnExecuteScalar" e em seu *Text* digite "ExecuteScalar". Dê um duplo clique no botão e inclua o código da **Listagem 2**.

Listagem 2. FbCommand com o ExecuteScalar

```
var
  command: FbCommand;
begin
  try try
      FbConnection1.Open;
      command := FbCommand.Create;
      command.Connection := FbConnection1;
      command.CommandText :=
      'SELECT COUNT(COUNTRY) FROM COUNTRY';
    Label1.Text := '<li>Quantidade de Países = ' +
      command.ExecuteScalar.ToString;
   except on E: Exception do
     Label1.Text := E.Message;
  end;
   finally
     FbConnection1.Close;
  end;
end;
```

Veja que, após a chamada do *Open*, estamos criando um objeto da classe *FbCommand*. Em seguida apontamos a propriedade *Connection* para o *FbConnection1* (poderíamos fazer isso dentro do *Create* do *FbCommand*). Configuramos a propriedade *CommandText* do *FbCommand* com um comando SQL que retorna a quantidade de registros da tabela *Country,* do *Employee.fdb*.

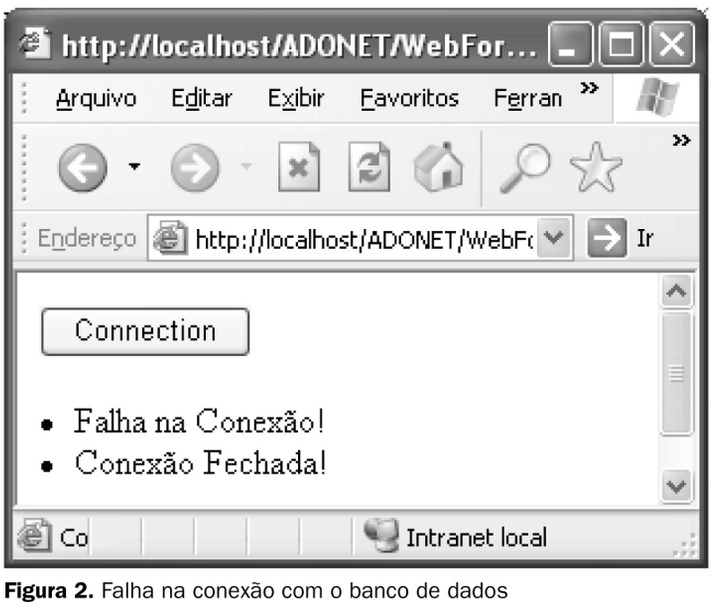


Figura 2. Falha na conexão com o banco de dados

Para finalizar, o resultado do *Select* é impresso no *Label*. Execute o projeto, clique no botão *ExecuteScalar* e confira o resultado na **Figura 3**.

Como você pode ver, o método *ExecuteScalar* é utilizado apenas para disparar comandos SELECT, e retornar um único valor. Muito útil quando queremos extrair valores simples, como um MAX, MIN, AVG ou o próprio COUNT que utilizamos no exemplo. Note que nesse exemplo, mudamos o bloco *try..except* para que pegue o valor da mensagem de erro, caso ela ocorra.

ExecuteReader e a classe FbDataReader

E para recuperarmos um conjunto de registros, como fazemos? Para isso utilizamos o *ExecuteReader* que retornará um objeto da classe *FbDataReader*. Um *FbDataReader* representa um conjunto de linhas resultante de um comando *Select*, porém ele permite apenas leitura e somente é possível mover seu cursor para frente.

Vamos fazer um teste? Inclua mais um botão na página, que deverá se chamar "btnExecuteReader", dê um duplo clique no mesmo e inclua o código da **Listagem 3**.

Listagem 3. FbCommand com o ExecuteReader

```
var
  command: FbCommand;
  reader: FbDataReader;
begin
  try try
      FbConnection1.Open();
      command := FbCommand.Create;
      command.Connection := FbConnection1;
      command.CommandText := 'SELECT * FROM COUNTRY';
      reader := command.ExecuteReader;
      while (reader.Read) do
    begin
      Label1.Text := Label1.Text + '<li><b>País: </b>'+
        reader['COUNTRY'].ToString + ' - ' +
        '<b>Moeda: </b>' +reader['CURRENCY'].ToString;
    end;
     reader.Close();
   except on E: Exception do
    Label1.Text := E.Message;
  end;
   finally
    FbConnection1.Close;
  end;
end;
```

Veja que agora, o *Select* não retornará apenas um valor e sim um conjunto de linhas. Observe também que estamos armazenando o retorno do *ExecuteReader* em um objeto da classe *FbDataReader*.

Através do *Read* podemos percorrer todas as linhas do *FbDataReader*. Faça um teste executando o projeto e clicando no botão *ExecuteReader* (**Figura 4**).

ExecuteNonQuery

No próximo exemplo vamos ver o funcionamento do *ExecuteNonQuery*. Esse método, como sugere seu nome, executará um comando SQL que não retornará valores ou linhas. O *ExecuteNonQuery* é utilizado para executar comandos como *Insert, Update* ou *Delete*. Inclua mais um botão na página, que deverá se chamar "btnExecuteNonQuery". Dê um duplo clique

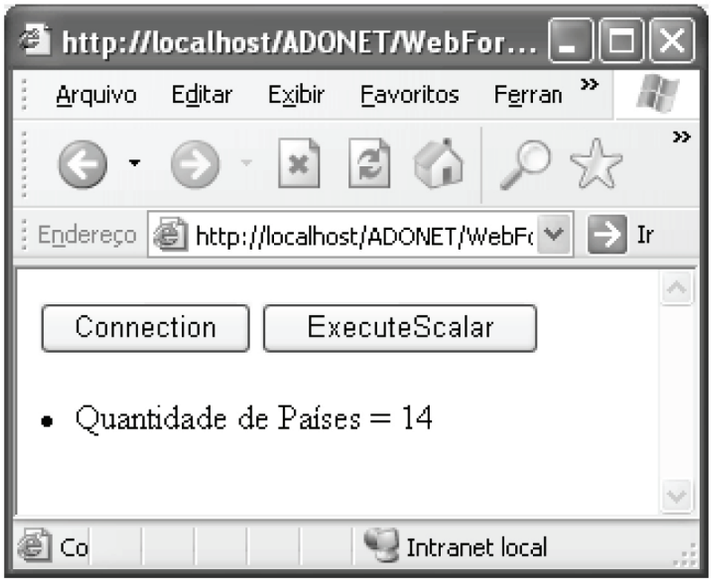


Figura 3. Método ExecuteScalar da classe SqlCommand

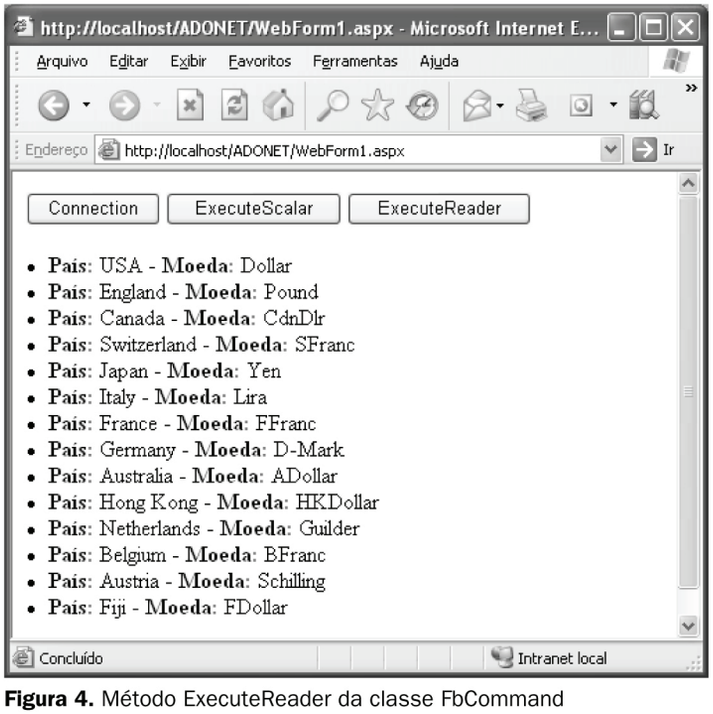


Figura 4. Método ExecuteReader da classe FbCommand

no mesmo e inclua o código da **Listagem 4**.

Listagem 4. FbCommand com o ExecuteNonQuery

```
var
  command: FbCommand;
begin
  try try
    FbConnection1.Open;
      command := FbCommand.Create;
      command.Connection := FbConnection1;
      command.CommandText :=
      'INSERT INTO COUNTRY VALUES(''Brasil'', ''real'')';
     command.ExecuteNonQuery;

     Label1.Text := '<li>Insert realizado com Sucesso!';
   except on E: Exception do
    Label1.Text := E.Message;
  end;
   finally
     FbConnection1.Close;

  end;
end;
```

Veja que estamos incluindo um comando *Insert* em *CommandText* e simplesmente imprimindo uma mensagem de conclusão. Caso o comando falhe, o bloco *try..except* será executado. Faça um teste executando a aplicação e clicando no botão *ExecuteNonQuery*.

Veja que a mensagem de conclusão foi impressa. Para conferir se o registro foi realmente incluso, clique no botão *ExecuteReader*, e veja como mostra a **Figura 5**, que o novo país foi realmente adicionado na tabela.

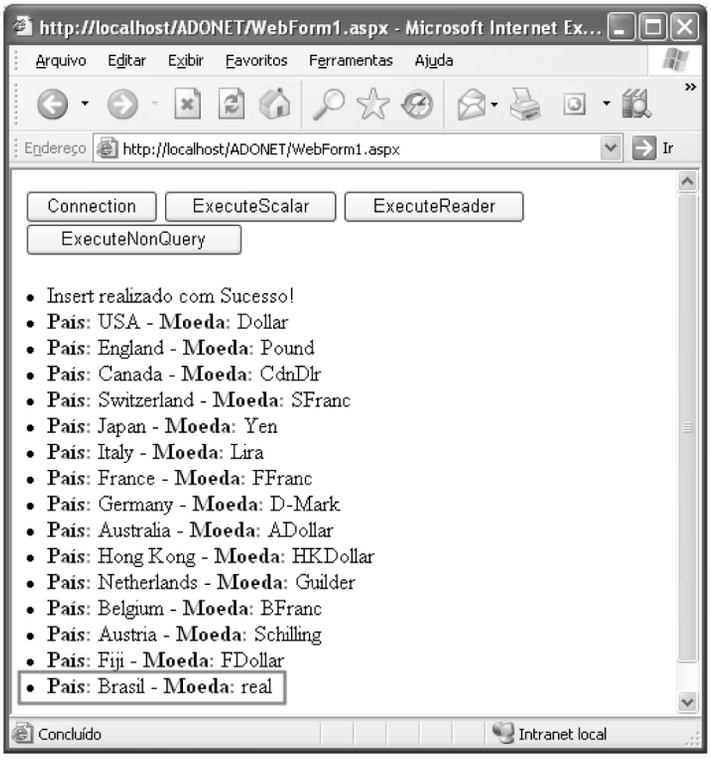


Figura 5. Resultado do método ExecuteNonQuery

DataSet e FbDataAdapter

Essas foram classes e métodos do grupo *Managed Providers*. Como vimos, com eles estamos sempre trabalhando com os dados de forma conectada ao banco. Agora veremos o grupo *Data Classes*, que permitem trabalhar de forma desconectada. A principal classe é o *DataSet*. Um *DataSet* é a representação em objetos do todo ou parte de um banco, tendo uma estrutura de classes que é muito semelhante à organização de um banco de dados.

O *DataSet* é composto de um ou mais *DataTables,* e pode conter relacionamentos entre esses, representados pelos *DataRelations*. Um *DataTable* por sua vez é composto de *DataColumns* e *DataRows*. Vamos direto à prática, que é a melhor forma de entendermos os *DataSets*.

Volte ao *design* da página *WebForm.aspx* e inclua mais um botão, que deverá se chamar "btnDataSet". A melhor forma de visualizarmos o funcionamento do *DataSet*, é com a utilização do *DataGrid*, para isso, arraste o mesmo na página. Dê um duplo clique sobre o botão que acabamos de adicionar e inclua o código da **Listagem 5**.

Listagem 5. Exemplo de utilização do DataSet e FbDataAdapter

```
var
  SelectString: string;
  adapter: FbDataAdapter;
  employee: DataSet;
begin
  SelectString :=
    'SELECT CUST_NO, CUSTOMER, CITY FROM CUSTOMER';
  adapter := FbDataAdapter.Create(SelectString,
    FbConnection1);
  employee := DataSet.Create;
  adapter.Fill(employee, 'CUSTOMER');
  DataGrid1.DataSource :=
    employee.Tables['CUSTOMER'].DefaultView;
  DataGrid1.DataBind();
end;
```

Observe no código da listagem anterior, que antes de mais nada estamos criando uma *string*, que contém o comando *Select* que retornará os registros da tabela *Customer* do banco. Um *DataSet* por si só, não recupera dados do banco, para isso precisamos de um *DataAdapter*. Como estamos trabalhando com o Firebird, utilizamos o *FbDataAdapter*. O *FbDataAdapter* pode usar o *FbConnection* como também a *string* de conexão passada ao construtor junto com o comando *Select*.

Em seguida, criamos um *DataSet* chamado *employee*, e através do método *Fill* do *FbDataAdapter* preenchemos o *DataSet* com o resultado do *Select* que foi definido. Em seguida estamos apontando a propriedade *DataSource* do *DataGrid* para a *DataTable Customer*, e chamando o *DataBind* do *DataGrid*, que mostrará os dados na página.

Vamos fazer um teste? Salve, compile e execute o projeto, e em seguida clique no botão *DataSet*. Veja que os registros da tabela *Customer* foram recuperados no *DataSet* e expostos no *DataGrid* (**Figura 6**).

Neste exemplo, o próprio *DataGrid* se encarregou de varrer as colunas e linhas do *DataTable* e apresentá-las na página. Também podemos varrer essas duas *Collections* se desejarmos. Inclua mais um botão em nossa interface, que deverá se chamar "btnDataSet2". Dê um duplo clique no botão e inclua o código da **Listagem 6**.

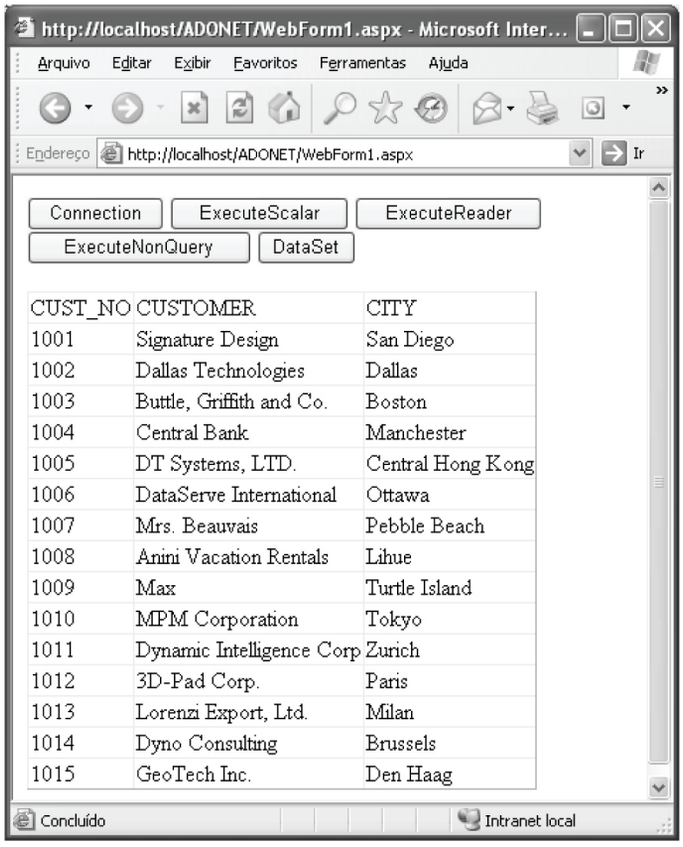


Figura 6. Exemplo da utilização do DataSet com o DataGrid

Listagem 6. Varrendo as Collections Rows e Columns

```
var
  SelectString: string;
  adapter: FbDataAdapter;
  employee: DataSet;
  row: DataRow;
  col: DataColumn;
begin
SelectString :=
  'SELECT CUST_NO, CUSTOMER, CITY FROM CUSTOMER';
adapter := FbDataAdapter.Create(SelectString,
  FbConnection1);
employee := DataSet.Create;
adapter.Fill(employee, 'Customer');
for row in employee.Tables['Customer'].Rows do
begin
  Label1.Text := Label1.Text + '<li>Customer: ';
  for col in employee.Tables['Customer'].Columns do
    Label1.Text := Label1.Text +
      row[col].ToString + ' - ';
  end;
end;
```

Veja que o procedimento de criação e preenchimento do *DataSet* é o mesmo. A diferença é que não estamos preenchendo o *DataGrid*. Através do uso da estrutura *for..in*, estamos varrendo as *Collections Rows* e *Colluns* do *DataTable*, e exibindo os registros no *Label*. Faça um teste executando o projeto (**Figura 7**).

Note que com a utilização dessas *Collections* podemos acessar qualquer atributo de qualquer linha da *DataTable*. Inclusive podemos alterar esses valores, de forma desconectada do banco, e depois dependendo da necessidade da aplicação, realizar a persistência dessas alterações, todas de uma vez.

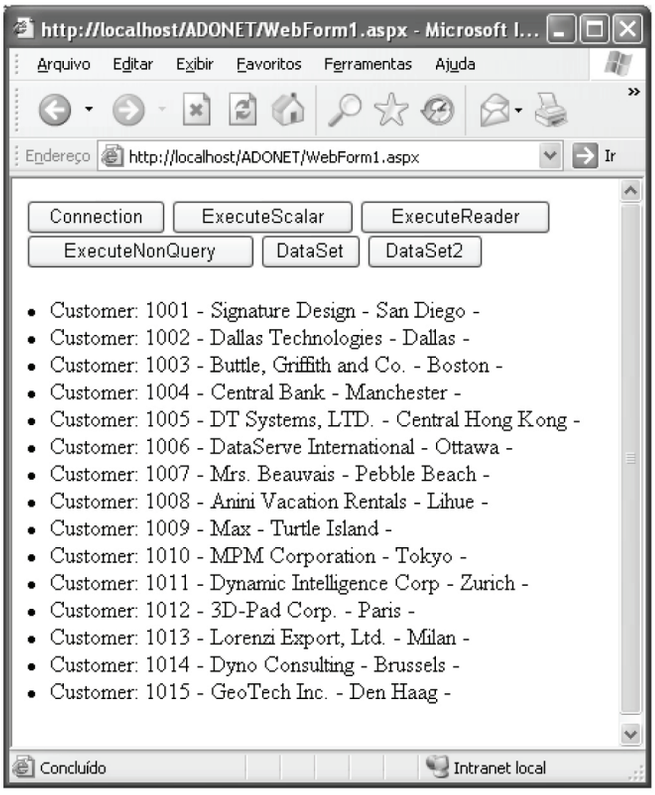


Figura 7. Exemplo da utilização do DataSet varrendo Rows e Collumns

Conclusão

Como pudemos ver, o ADO.NET é uma biblioteca completa de acesso a banco de dados. Os exemplos vistos neste artigo utilizam o Firebird, mas podemos facilmente realizar o mesmo acesso à Oracle ou qualquer outro banco usando *OleDb* ou ODBC, suportados pelo ADO.NET.

Um ponto muito importante a ser observado é que em todos os testes realizados aqui, a *string* de conexão está no editor do *FbConnection*. O correto seria armazenar essa informação em um arquivo de configuração (*Web.config*) do projeto. Para obtermos um nível de segurança maior, podemos até criptografar essa informação.

Outra observação muito importante é quanto ao acesso desconectado do banco. Lembre-se que em páginas Web estamos sempre sujeitos aos temíveis *postbacks*. Toda vez que um *postback* ocorre, perdemos as informações armazenadas nos *DataSets*.

Para evitar uma consulta ao banco sempre que houver um *postback*, utilize as variáveis de sessão disponíveis em aplicações ASP.NET. Espero que este artigo tenha trazido clareza sobre o ADO.NET, que sem dúvida é uma biblioteca muito simples e fácil de se utilizar.

Grande abraço a todos e até a próxima! ■

CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como trabalhar com Firebird no ASP.NET.

www.devmedia.com.br/articles/viewcomp.asp?comp=561

CLUBEDELPHI PLUS!

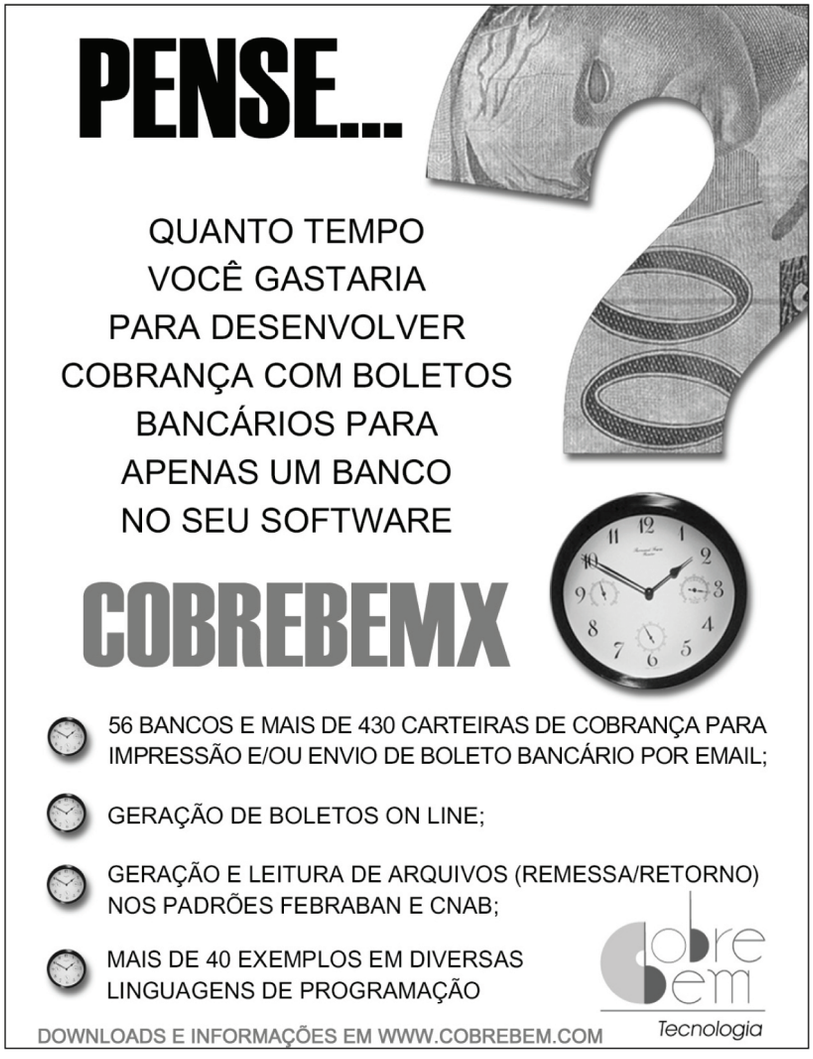
Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como trabalhar com *Stored Procedures* no ADO.NET.

www.devmedia.com.br/articles/viewcomp.asp?comp=564

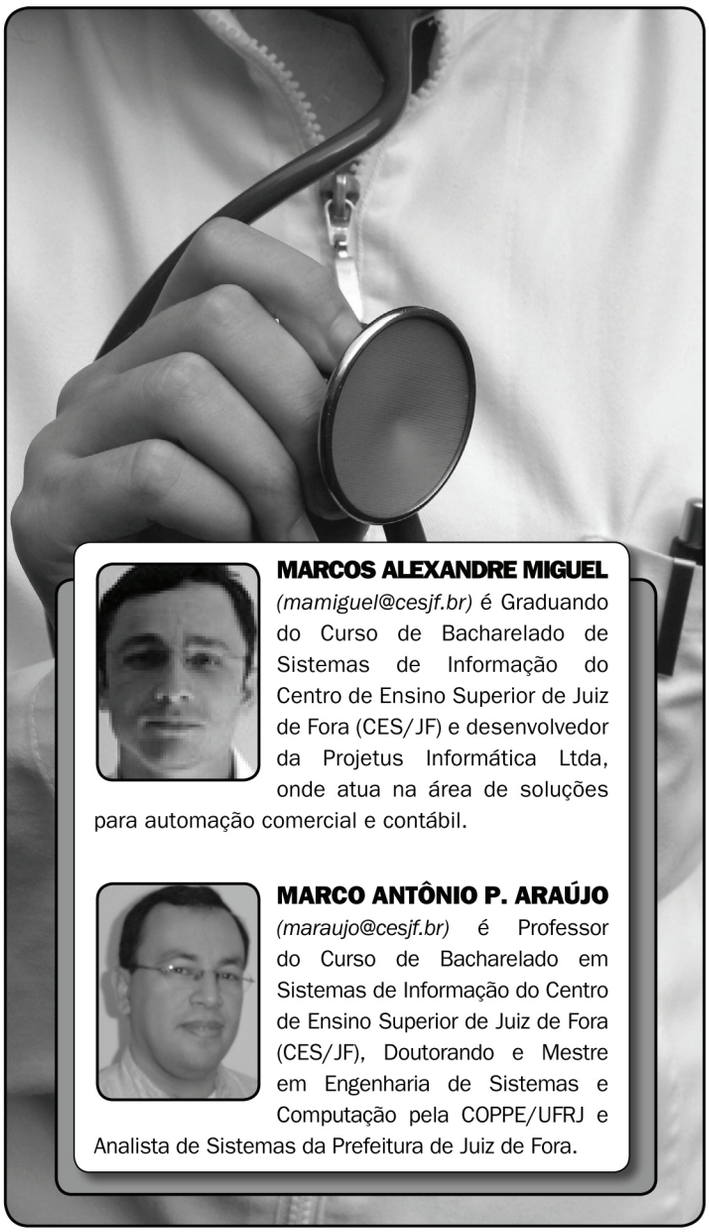
CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como trabalhar com várias Views no ADO.NET com o componente DataSet.

www.devmedia.com.br/articles/viewcomp.asp?comp=560



QA Audits - faça um ckeck-up no seu código-fonte



MARCOS ALEXANDRE MIGUEL *(mamiguel@cesjf.br)* é Graduando do Curso de Bacharelado de Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora (CES/JF) e desenvolvedor da Projetus Informática Ltda, onde atua na área de soluções para automação comercial e contábil.

MARCO ANTÔNIO P. ARAÚJO *(maraujo@cesjf.br)* é Professor do Curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora (CES/JF), Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ e Analista de Sistemas da Prefeitura de Juiz de Fora.

Auditorias são técnicas que buscam verificar se artefatos de software (como especificações, modelos e código-fonte) estão em conformidade com os padrões definidos e em uso na instituição. Auditorias no código-fonte são as mais conhecidas e, quando aplicadas manualmente, geralmente consistem na seleção aleatória de alguns programas do sistema, objetivando verificar se atendem aos padrões definidos.

A importância em obedecer a esses padrões é que tornam os programas mais fáceis de entender e modificar, melhorando a qualidade interna dos sistemas e proporcionando uma maior produtividade por parte das equipes de desenvolvimento e também de manutenção.

O Delphi 2006 trouxe uma ferramenta chamada QA Audits integrada ao IDE para realizar auditorias de software a partir do código-fonte, sugerindo uma série de padrões de codificação que representam boas técnicas de programação. Essa ferramenta avalia a conformidade do código-fonte com as auditorias selecionadas e, através dos resultados gerados, pode-se verificar se o código está aderente aos padrões de codificação em uso, podendo indicar partes que necessitam ser reestruturadas.

Este artigo apresenta a ferramenta QA Audits através de um estudo de caso, considerando inicialmente um trecho de código contendo algumas violações das auditorias sugeridas pela ferramenta e, em seguida, como o código deveria ser modificado de forma a atender os padrões estabelecidos por essas auditorias.

A ferramenta QA Audits

Para utilizar a ferramenta QA Audits em um projeto existente, deve-se acessar a guia *Model View* do *Project Manager* (**Figura 1**).

Será apresentada uma mensagem perguntando se o usuário deseja configurar o projeto para utilizar suporte à modelagem. Confirmando a pergunta, a ferramenta de auditoria estará disponível através do menu de contexto (**Figura 2**).

Quando se executa a ferramenta, verifica-se um grande número de opções de auditorias para serem escolhidas, agrupadas em diversas categorias. Cada opção de auditoria contém descrições sobre os padrões de codificação que serão verificados, como também atribui um grau de severidade para a violação da auditoria.

As descrições das auditorias suportadas pelo Delphi 2006 estão disponíveis na janela principal da ferramenta QA Audits (**Figura 3**).

No painel inferior dessa janela é mostrada a definição de uma determinada auditoria, bem como, em muitos casos, um exemplo incorreto e outro correto do código-fonte relacionados à auditoria. Essa janela ainda pode ser utilizada para salvar ou carregar auditorias dos projetos, bem como configurar quais auditorias serão aplicadas.

Os botões da esquerda para a direita têm as seguintes funções:

Load Set: Abre um arquivo ADT com auditorias customizadas salvas, facilitando a utilização de auditorias já configuradas;

Figura 1. Project Manager

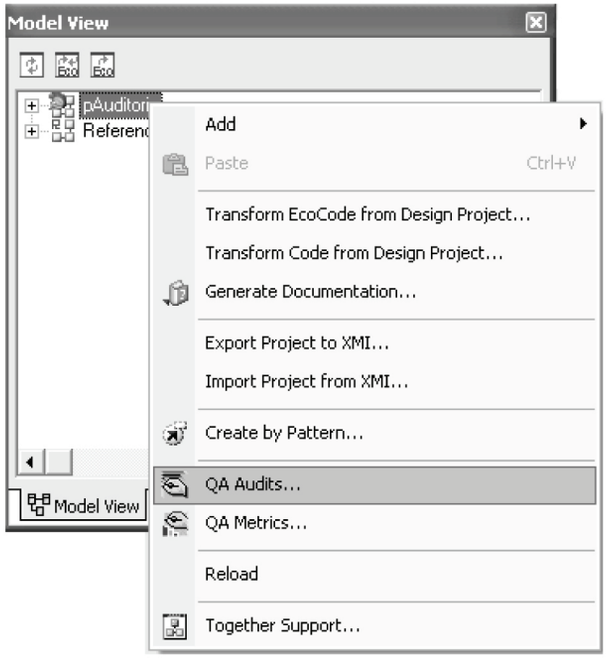


Figura 2. Acesso a auditoria de software suportadas pelo Delphi 2006

Save Set As: Abre uma caixa de diálogo permitindo salvar auditorias configuradas, podendo ser usadas novamente;

Select All: Marca todos os *CheckBoxes* de auditoria;

Unselect All: Desmarca todos os *CheckBoxes* de auditoria;

Set Defaults: Retorna ao conjunto de auditorias padrão;

Find Audit: Navega através das auditorias pesquisando pela abreviatura de uma auditoria, através de uma caixa de diálogo.

A opção *Scope* indica as partes do projeto que a auditoria será aplicada. A seleção permite escolher se os exames serão processados para o modelo, uma classe ou o projeto inteiro, selecionado antes de invocar as verificações da auditoria.

Existe ainda um painel de opções à direita da janela que varia dependendo da auditoria selecionada, tendo as seguintes opções:

Categorized: Organiza as propriedades da auditoria em grupos expansíveis;

Alphabetic: Organiza as propriedades da auditoria em ordem alfabética.

O item *Severity* (**Figura 4**) vem com um valor padrão de severidade definido, podendo-se atribuir os seguintes valores para esse campo: *Info*, *Warning*, *Error* e *Fatal*, definindo o quão sério é a violação de uma auditoria, indicada nos resultados gerados pela verificação das auditorias.

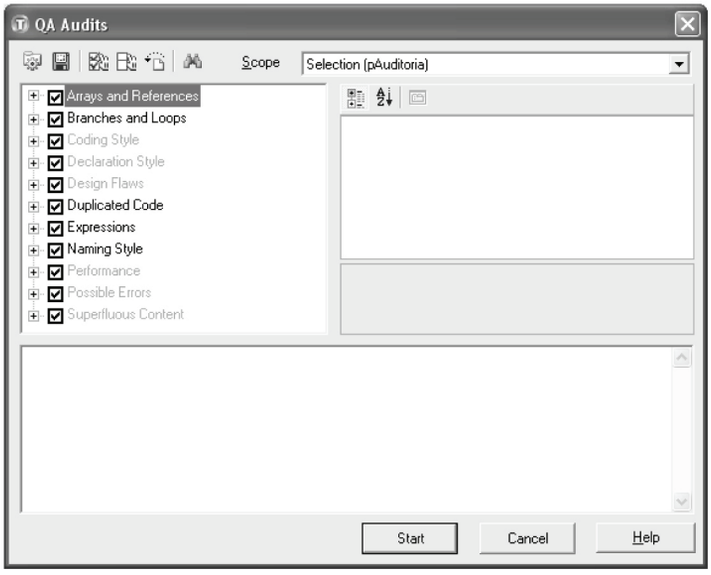


Figura 3. Auditorias suportadas pelo Delphi 2006

Os resultados da execução da auditoria indicam somente as violações dos padrões estabelecidos de modo que se possa examinar cada situação e decidir se o código-fonte deve ser modificado.

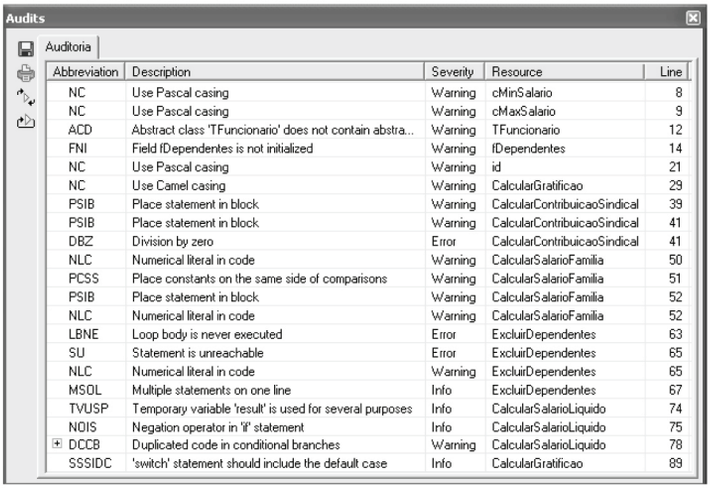
Figura 4. Definição da severidade de uma auditoria

Estudo de Caso

Com o intuito de demonstrar a utilização desses recursos, será apresentado um estudo de caso que abordará algumas auditorias de software aplicadas em um projeto. A **Listagem 1** contém uma série de violações que serão avaliadas pelas auditorias existentes.

Os padrões definidos podem ser usados como convenção de boas práticas de codificação, avaliando a aderência do código-fonte aos padrões previamente definidos. O código apresentado na **Listagem 1** mostrará os seguintes resultados de auditoria (**Figura 5**), após clicar no botão *Start* na janela principal (mostrada na **Figura 3**) da ferramenta QA Audits.

A análise do resultado da auditoria mostra a abreviatura, descrição e a severidade de cada auditoria violada, bem como a linha onde se encontra a violação dentro do código-fonte. A seguir, veremos cada uma dessas auditorias apresentadas, as linhas de código onde se encontram tais violações, bem como uma possível correção para as violações apresentadas.



Abbreviation	Description	Severity	Resource	Line
NC	Use Pascal casing	Warning	cMinSalario	8
NC	Use Pascal casing	Warning	cMaxSalario	9
ACD	Abstract class 'TFuncionario' does not contain abstra...	Warning	TFuncionario	12
FNI	Field fDependentes is not initialized	Warning	fDependentes	14
NC	Use Pascal casing	Warning	id	21
NC	Use Camel casing	Warning	CalcularGratificao	29
PSIB	Place statement in block	Warning	CalcularContribuicaoSindical	39
PSIB	Place statement in block	Warning	CalcularContribuicaoSindical	41
DBZ	Division by zero	Error	CalcularContribuicaoSindical	41
NLC	Numerical literal in code	Warning	CalcularSalarioFamilia	50
PCSS	Place constants on the same side of comparisons	Warning	CalcularSalarioFamilia	51
PSIB	Place statement in block	Warning	CalcularSalarioFamilia	52
NLC	Numerical literal in code	Warning	CalcularSalarioFamilia	52
LBNE	Loop body is never executed	Error	ExcluirDependentes	63
SU	Statement is unreachable	Error	ExcluirDependentes	65
NLC	Numerical literal in code	Warning	ExcluirDependentes	65
MSOL	Multiple statements on one line	Info	ExcluirDependentes	67
TVUSP	Temporary variable 'result' is used for several purposes	Info	CalcularSalarioLiquido	74
NOIS	Negation operator in 'if' statement	Info	CalcularSalarioLiquido	75
DCCB	Duplicated code in conditional branches	Warning	CalcularSalarioLiquido	78
SSSIDC	'switch' statement should include the default case	Info	CalcularGratificao	89

Figura 5. Listagem da auditoria feita pela ferramenta QA Audits

Listagem 1. Listagem do código-fonte com violações de auditorias

```
01: unit UntFuncionario;
02:
03: interface
04:
05: uses Classes, SysUtils, DateUtils;
06:
07: const
08:   cMinSalario: Extended = 3500;
09:   cMaxSalario: Extended = 350;
10:
11: type
12:   TFuncionario = class abstract
13:   private
14:     fDependentes: TStringList;
15:     fSalario: Currency;
16:     fSindicalizado: Boolean;
17:     fid: Integer;
18:     fAdicionalNoturno: Boolean;
19:     fNome: string;
20:
21:    property id: Integer read fid write fid;
22:     property Nome: String read fNome write fNome;
23:     property Salario: Currency read fSalario write fSalario;
24:     property AdicionalNoturno: Boolean
          read fAdicionalNoturno write fAdicionalNoturno;
25:     property Sindicalizado: Boolean
          read fSindicalizado write fSindicalizado;
26:     function CalcularSalarioLiquido: Extended;
27:     function CalcularSalarioFamilia: Extended;
28:     function CalcularContribuicaoSindical(
          diasMesCorrente: Integer): Extended;
29:     function CalcularGratificao(DiasTrabalhados: Integer): Extended;
30:     procedure ExcluirDependentes;
31:   public
32:   end;
33:
34: implementation
35:
36: function TFuncionario.CalcularContribuicaoSindical(
      DiasMesCorrente: Integer): Extended;
37: begin
38:   if DiasMesCorrente <> 0 then
39:     Result:= 0
40:   else if fSindicalizado then
41:     Result:= fSalario / DiasMesCorrente;
42: end;
43:
44: function TFuncionario.calcularSalarioFamilia: Extended;
45: var
46:   i: Integer;
47: begin
48:    for i:= 0 to fDependentes.Count - 1 do
49:    begin
50:     if (YearsBetween(Date, StrToDate(fDependentes[i])) < 14) and
51:       (fSalario <= cMaxSalario) and (cMinSalario >= fSalario) then
52:       Result:= Result + 13.48;
53:    end;
54: end;
55:
56: procedure TFuncionario.ExcluirDependentes;
57: var
58:  limite: Boolean;
59:   indice: Integer;
60: begin
61:   limite:= False;
62:
63:   while limite do
64:   begin
65:    if YearsBetween(Date, StrToDate(fDependentes[indice])) >= 14 then
66:    begin
67:       fDependentes.Delete(indice); exit;
68:    end;
69:    Inc(indice);
70:   end;
71: end;
72:
73: function TFuncionario.calcularSalarioLiquido: Extended;
74: begin
75:  if not fAdicionalNoturno then
76:   begin
77:     Result:= fSalario;
78:     Result:= Result + CalcularGratificao(30) +
          CalcularSalarioFamilia();
79:   end
80:   else
81:   begin
82:     Result:= (fSalario * 1.20);
83:     Result:= Result + CalcularGratificao(30) +
          CalcularSalarioFamilia();
84:   end;
85: end;
86:
87: function TFuncionario.calcularGratificao(
      DiasTrabalhados: Integer): Extended;
88: begin
89:  case DiasTrabalhados of
90:     25..31: Result:= fSalario * 1.30;
91:     19..24: Result:= fSalario * 1.15;
92:   end;
93: end;
94:
95: end.
```

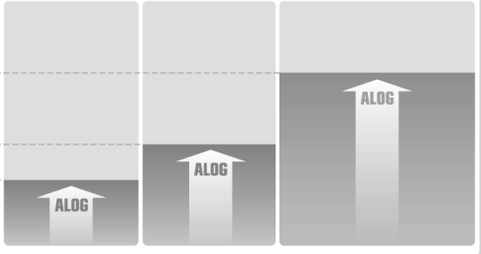
FAÇA COMO A ALOG
FOQUE EM SEU NEGÓCIO

A ALOG é totalmente focada em serviços de Data Center. Dedica-se 24 horas por dia, 7 dias por semana a cuidar do negócio dos nossos 320 clientes corporativos.

ALOG desenvolveu no Brasil o conceito de **HOSTING GERENCIADO**: um serviço de excelência que garantirá a sua empresa estabilidade, agilidade, redundância, e escalabilidade. Para que isso seja verdade os clientes da ALOG contam com **Suporte Incondicional***, um conceito revolucionário em atendimento no Brasil.

ENTENDA O **HOSTING GERENCIADO**

CAMADAS DE NEGÓCIO	DESCRIÇÃO
Processo de Negócio	Atividades cotidianas do cliente
Aplicação do Cliente	Softwares de Gestão (Ex: ERP, CRM, outros)
Ambiente da Aplicação	.NET, JAVA, SQL Server, MySQL, Oracle, Exchange
Sistema Operacional	Windows, Linux, Unix, Solaris
Servidor ou equipamento	Servidores RISC/CISC, Appliances, Roteadores, Switches
Rede	IP, Wan, Rede Local
Infra-estrutura Básica	Energia, HVAC, Segurança - 2N+1



ESTRATÉGIA DO CLIENTE	COLOCATION	HOSTING	HOSTING GERENCIADO

PLANOS DE **HOSTING GERENCIADO**

GERENCIADO 1

- **Pentium D 2.8 GHz**
- 512 RAM DDR2
- 2 HD SATA 80 GB
- 450 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$**1.050,00** mensais

INSTALAÇÃO IMEDIATA***

GERENCIADO 2

- **Dual Xeon Dual Core 3.0 GHz**
- 1GB RAM DDR2 533
- 2 HD SATA RAID I 300 GB
- 1.000 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$**2.850,00** mensais

INSTALAÇÃO IMEDIATA***

* Sujeito a condições contratuais ** Podendo escolher: HD, RD e GT *** Sujeito a disponibilidade no estoque

ALOG data centers

Excelência em Hosting
www.alog.com.br :: 0800 282 3330

Hosting Gerenciado I Colocation I Contingência I Email Corporativo I Conectividade I Serviços Profissionais

Convenções de nomeação (NC - Naming Conventions)

A auditoria NC verifica se o código está de acordo com os seguintes padrões de nomeação:

- Um nome de interface deve começar com o prefixo "I";
- O primeiro caractere de cada palavra deve ser maiúsculo quando usado em classes, constantes, tipos enumerados, valores enumerados, eventos, interfaces, métodos, *namespaces*, propriedades, atributos estáticos e nomes de campos públicos;
- O primeiro caractere de cada palavra, *exceto* a primeira, deve ser maiúsculo quando usada em variáveis locais, parâmetros formais e nomes de campos não públicos;
- Nomes de classes de exceção devem ter terminação *Exception*.

A análise do resultado da auditoria aponta para as linhas 8, 9, 21 e 29 do código-fonte onde se encontram as violações. Pode-se executar a correção dessas violações seguindo as convenções descritas, capitalizando a primeira letra das constantes dos campos e o parâmetro das funções. A **Listagem 2** mostra a correção dessas violações.

Listagem 2. Correção dos problemas encontrados pela auditoria NC

```
const
  CMinSalario: Extended = 3500;
  CMaxSalario: Extended = 350;
....
public
   property Id: Integer read fId write fId;
....
   function CalcularGratificao(
     diasTrabalhados: Integer): Extended;
....
```

Declaração de classe abstrata (ACD - Abstract Class Declaration)

A auditoria ACD verifica declarações de classes abstratas, emitindo uma mensagem de advertência se uma classe abstrata não contém métodos abstratos. Nesse caso, a classe deve ser convertida em uma classe não abstrata ou então tornar seus métodos públicos, abstratos e reutilizáveis por suas subclasses.

A análise do resultado da auditoria aponta para a linha 12 do código-fonte onde se encontra a violação. Uma possível correção para esse tipo de problema seria retirar a definição da classe como abstrata, conforme o código apresentado na **Listagem 3**.

Listagem 3. Correção do problema encontrado pela auditoria ACD

```
type
  TFuncionario = class
```

Declaração de classe restrita (RCD - Restricted Class Declaration)

A auditoria RCD detecta as classes que não podem ser usadas por seus clientes. Essas classes podem ser:

- Classes públicas que não contêm membros e construtores públicos ou protegidos. Apesar do acesso público, tais classes não podem ser usadas por outras classes;
- Classes não públicas que contêm construtores públicos também não podem ser utilizadas por outras classes.

Após a modificação sugerida na auditoria anterior (ACD), uma nova execução do conjunto de auditorias mostrará a indicação de violação da auditoria RCD, conforme a **Figura 6.**

A análise do resultado dessa auditoria aponta para a linha 12 do código-fonte onde se encontra a violação. Uma possível correção para esse tipo de problema seria reposicionar os métodos que podem ser colocados na seção *public* da classe, conforme a **Listagem 4**.

Listagem 4. Correção do problema encontrado pela auditoria RCD

```
type
  TFuncionario = class
private
...
public
...
  function CalcularSalarioLiquido: Extended;
  function CalcularSalarioFamilia: Extended;
  function CalcularContribuicaoSindical(
    diasMesCorrente: Integer): Extended;
...
end;
```

Campo não inicializado (FNI - Field Not Initialized)

A auditoria FNI detecta se existem campos da classe que não foram inicializados, em três situações diferentes: na definição da classe, no método construtor ou em alguma indicação de atribuição.

A análise do resultado da auditoria aponta para a linha 14 do código-fonte que contém um campo *fDependentes* com a

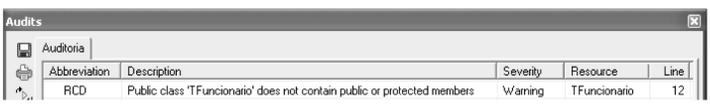


Figura 6. Violação da auditoria RCD

lista dos dependentes do funcionário, mas a mesma não foi inicializada. Para solucionar esse problema, o método construtor da classe funcionário poderia inicializar o campo.

Esse objeto criado deve ser destruído quando o mesmo não necessitar ser mais ser usado. Para assegurar isso, pode-se implementar os métodos construtor e destrutor da classe, conforme a **Listagem 5**.

Listagem 5. Correção do problema encontrado pela auditoria FNI

```
constructor TFuncionario.Create();
begin
  fDependentes:= TStringList.Create;
end;

destructor TFuncionario.Destroy;
begin
  if Assigned(fDependentes) then
  begin
    fDependentes.Free;
  end;
end;
```

Colocar Instrução em bloco (PSIB - Place Statement In Block)

A auditoria PSIB detecta se um laço contém um bloco (*begin/end*) ou se as partes *then* e *else* de uma instrução *if* estão definidas através de blocos. Isso torna mais fácil adicionar novas instruções sem acidentalmente introduzir defeitos em futuras manutenções.

A análise do resultado da auditoria aponta para as linhas 39 e 41 do código-fonte onde se encontram as violações. Uma possível correção para esse tipo de problema seria adicionar blocos *begin/end*, conforme a **Listagem 6.**

Listagem 6. Correção do problema encontrado pela auditoria PSIB

```
if DiasMesCorrente <> 0 then
begin
  Result:= 0;
end
else if fSindicalizado then
begin
  Result:= fSalario/DiasMesCorrente;
end;
```

Divisão por zero (DBZ - Division By Zero)

A auditoria DBZ detecta as situações em que um determinado cálculo pode gerar uma divisão por zero, resultando em um *ArithmeticException* em tempo de execução. A análise do resultado da auditoria aponta para a linha 41 do código-fonte onde se encontra o problema detectado.

Uma possível correção para esse tipo de problema seria validar se o divisor da expressão pode ser zero antes de executar o cálculo. A **Listagem 7** apresenta a correção e uma melhoria na formulação do algoritmo, tornando-o mais legível e menos suscetível a este problema.

Listagem 7. Correção do problema encontrado pela auditoria DBZ

```
Result := 0;
if (DiasMesCorrente <> 0) and fSindicalizado then
begin
  Result:= fSalario/DiasMesCorrente;
end;
```

Colocar constantes no mesmo lado de comparações (PCSS - Place Constants on the Same Side of Comparisons)

A auditoria PCSS é utilizada quando um objeto é comparado a uma constante, podendo ser colocado à esquerda ou à direita do operador na comparação. A opção *Side* (**Figura 7**) permite selecionar se as constantes devem ser colocadas à direita ou à esquerda do operador na comparação.

A análise do resultado da auditoria aponta para a linha 51 do código-fonte onde se

Livros de Delphi

Delphi - Implementação de Algoritmos e Técnicas p/ Ambientes Visuais
Visual Books
R$ 62,00

Delphi: para Todas as Versões
Ciência Moderna

R$ 29,00

Estudo Dirigido de Delphi 2005
Érica
R$ 78,50

Delphi 2005: Aplicações de Banco de Dados com InterBase 7.5 e MySQL 4.0.23
Érica

R$ 149,50

Introdução ao Desenvolvimento de Aplicações em Delphi
Uniderp
R$ 50,00

O Melhor da Delphi Line (CD 1 e 2)
Sistemas e Exemplos com fontes e Delphi
Ramos da Informática

10% desconto
De: R$ 134,00 Por: R$ 120,00

Delphi e Relatórios - Treinamento a Distancia (CD)2005
Clube Delphi
10% desconto
De: R$ 200,00 Por: R$ 180,00

Guia do Desenvolvedor de Delphi for .NET
Makron Books

10% desconto
De: R$ 173,50 Por: R$ 156,15

Delphi 8 Para Plataforma .NET: Curso Completo
Axcel
10% desconto
De: R$ 249,00 Por: R$ 224,10

Sistema Comercial Integrado com Delphi 2005: Cadastro e Estoque
Érica

10% desconto
De: R$ 135,00 Por: R$ 121,50

Frete Grátis
Para todo o Brasil!
3x no Visa e Mastercard
e 6x no Hipercard

Outras facilidades de pagamento:

livrosdeprogramacao
.com.br

encontra a violação. Uma possível correção para esse tipo de violação é fazer com que as constantes estejam do mesmo lado das condições, conforme o código apresentado pela **Listagem 8**.

Listagem 8. Correção do problema encontrado pela auditoria PCSS

```
for i:= 0 to fDependentes.Count - 1 do
begin
  if (YearsBetween(Date, StrToDate(fDependentes[i])) < 14) and
    (fSalario <= CMaxSalario) and (fSalario >= CMinSalario) then
  begin
    Result:= Result + 13.48;
  end;
end;
```

Literal numérico no código (NLC - Numerical Literal in Code)

A auditoria NCL sugere substituir números literais por constantes simbólicas. Nessa auditoria não são avaliados os valores fixos "-1", "0", e "1". A análise do resultado da auditoria aponta para a linha 52 do código-fonte onde se encontra a violação.

Uma possível correção para esse tipo de problema seria adicionar no código uma constante que tenha o valor da cota de salário família ou então uma função que resulte nesse valor. Nesse caso, essa correção se torna uma boa prática de programação, porque torna o valor reutilizável e de fácil manutenibilidade. O resultado após a correção é mostrado na **Listagem 9**.

Listagem 9. Correção do problema encontrado pela auditoria NLC

```
...
const
  CCotaSalarioFamilia: Extended = 13.48;
...
for i:= 0 to fDependentes.Count - 1 do
begin
    if (YearsBetween(Date, StrToDate(fDependentes[i])) < 14) and
      (fSalario <= cMaxSalario) and (cMinSalario >= fSalario) then
    begin
      Result:= Result + CCotaSalarioFamilia;
    end;
end;
```

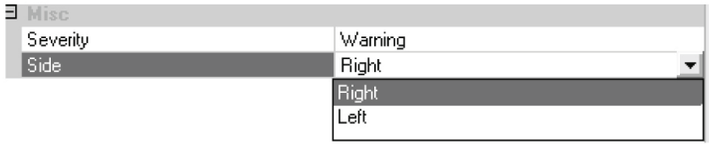


Figura 7. Opções da auditoria PCSS

Código do laço nunca é executado (LBNE - Loop Body is Never Executed)

A auditoria LBNE detecta situações onde a condição *repeat*, *while* ou *for* de um laço é sempre falsa, assim o código no "corpo" do laço nunca será executado. A análise do resultado da auditoria aponta para a linha 63 do código-fonte onde se encontra a violação.

Uma possível correção para esse tipo de problema seria substituir a condição de verificação do *while* por outro tipo de verificação, conforme a **Listagem 10**.

Listagem 10. Correção do problema encontrado pela auditoria LBNE

```
indice := 0;
while indice < fDependentes.Count do
begin
  if YearsBetween(Date, StrToDate(
    fDependentes[indice])) >= 14 then
  begin
    fDependentes.Delete(indice); exit;
  end;
Inc(indice);
end;
```

Instrução inacessível (SU - Statement is Unreachable)

A auditoria SU relata para uma instrução que nunca será executada. A mensagem de advertência é gerada uma vez para cada instrução não usada. A análise do resultado da auditoria aponta para a linha 65 do código-fonte onde se encontra a violação.

Uma possível correção para esse tipo de problema é fazer com que a instrução *while* desse bloco de código seja executada, fazendo assim com que condição dentro do laço também seja executada. A correção da auditoria anterior (**Listagem 10**) ajustará essa violação também.

Múltiplas instruções em uma linha (MSOL - Multiple Statements on One Line)

A auditoria MSOL certifica-se de que cada linha de código contenha apenas uma instrução. A análise do resultado da auditoria aponta para a linha 67 do código-fonte onde se encontra a violação. Pode-se ajustar essa violação por colocar cada instrução em uma linha específica, conforme apresentado na **Listagem 11**.

Listagem 11. Correção do problema encontrado pela auditoria MSOL.

```
if YearsBetween(Date, StrToDate(fDependentes[indice])) >= 14 then
begin
  fDependentes.Delete(indice);
  Exit; <-
end;
Inc(indice);
```

Operador de negação em instrução if (NOIS - Negation Operator in If Statement)

A auditoria NOIS indica que o operador de negação em instruções condicionais pode atrapalhar a legibilidade do programa, assim recomenda-se que não deve ser usado freqüentemente. A análise do resultado da auditoria aponta para a linha 75 do código-fonte onde se encontra a violação.

Pode-se ajustar essa violação por inverter as condições de verificação conforme o trecho de código apresentado na **Listagem 12**.

Listagem 12. Correção do problema encontrado pela auditoria NOIS

```
if fAdicionalNoturno then
begin
  Result:= fSalario * 1.20;
  ...
end
else
begin
   Result:= fSalario;
   ...
end;
```

Código duplicado em ramos condicionais (DCCB - Duplicated Code in Conditional Branches)

A auditoria DCCB verifica instruções condicionais onde suas possibilidades, verdadeira e falsa, têm código comum. Tal código deveria ser movido para fora das estruturas condicionais. A análise do resultado da auditoria aponta para a linha 78 do código-fonte onde se encontra a violação.

Além de mostrar em qual linha se encontra o problema ao se expandir a auditoria, pode-se analisar quais fragmentos de código estão em duplicidade.

Para corrigir esse tipo de violação, deve-se mover para fora do bloco condicional as declarações que estão duplicadas conforme pode-se verificar na **Listagem 13**.

Listagem 13. Correção do problema encontrado pela auditoria DCCB

```
Result := fSalario;
if fAdicionalNoturno then
begin
   Result := Result * 1.20;
end;
Result:= Result + CalcularGratificao(30) + CalcularSalarioFamilia();
```

O código também foi modificado para uma melhor legibilidade do mesmo.

Variável temporária usada para vários propósitos (TVUSP - Temporary Variable is Used for Several Purposes)

A auditoria TVUSP descobre se a mesma variável temporária está sendo usada para vários propósitos diferentes. A análise do resultado da auditoria aponta para a linha 74 do código-fonte onde se encontra a violação.

Para resolver essa violação deve-se dividir a utilização da variável temporária. O trecho de código apresentado nessa auditoria utiliza a variável *Result* da função para efetuar todos os cálculos do salário do funcionário.

Uma possível correção seria criar uma variável temporária para armazenar o percentual e depois multiplicá-lo uma única vez para obter o salário com o valor do adicional noturno, conforme **Listagem 14**.

Listagem 14. Correção do problema encontrado pela auditoria TVUSP

```
var
  Percentual: Extended;
begin
  Percentual := 1.00;
  if fAdicionalNoturno then
  begin
    Percentual := 1.20;
  end;
  Result := (fSalario * Percentual) +
    CalcularGratificao(30) + CalcularSalarioFamilia();
end;
```

Instrução Switch deve incluir valor default (SSSIDC - Switch Statement Should Include the Default Case)

A auditoria SSSIDC certifica-se de que cada instrução *case* tenha um valor *default*. A análise do resultado da auditoria aponta para a linha 89 do código-fonte onde se encontra o problema detectado.

Uma possível correção para esse tipo de problema é colocar na instrução *case* um operador condicional *else* para os valores que não estão nas faixas de verificação, conforme a **Listagem 15**.

Listagem 15. Correção do problema encontrado pela auditoria SSSIDC.

```
case diasTrabalhados of
  25..31: Result:= fSalario * 1.30;
  19..24: Result:= fSalario * 1.15;
else
  Result:= 0; <-
end;
```

Conclusão

O objetivo deste artigo foi mostrar um novo e interessante recurso do Delphi 2006 que aplica auditorias para auxiliar na melhoria da qualidade e legibilidade do código-fonte. As auditorias apresentadas neste artigo são apenas algumas dentro do catálogo disponível aos desenvolvedores e suas equipes e tendem a tornar o código padronizado, obedecendo às regras que podem melhorar a qualidade do software. ■

Validação simplificada com os controles da JVCL

FABRÍCIO DESBESSEL
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site http://www.fabricio.pro.br.

A importância de validarmos os dados inseridos em nossas aplicações já foi amplamente discutida em artigos de outras edições. Mais importante que discutir, é mostrar formas de validar a entrada de dados nas suas aplicações. Quem já desenvolveu em Delphi para ASP.NET deve ter utilizado os componentes de validações, que são bem fáceis de usar e solucionam muito bem a questão.

A boa notícia é que para aplicações VCL temos a mesma possibilidade, através de componentes semelhantes. Trata-se de componentes presentes na biblioteca JEDI VCL, que é uma comunidade internacional de voluntários que desenvolvem utilitários, componentes e bibliotecas para o Delphi, isso tudo *open source*.

Instalação do JEDI VCL

Primeiramente, precisamos baixar a biblioteca do site do projeto JEDI. Acesse: *sourceforge.net/project/showfiles.php?group_id=45786* e baixe o arquivo JVCL 3.20 ou uma versão mais recente. Opte sempre pelo último *build* e pela versão completa, pois os componentes da JVCL precisam da JCL (biblioteca de códigos).

Após baixar o arquivo, descompacte e execute o arquivo *Install.bat* que encontra-se dentro da pasta *jcl*. Siga todos os passos e finalize a instalação. Agora execute o *Install.bat* da pasta *jvcl* seguindo todos os passos até a finalização. Com isso teremos instalado os componentes da JVCL e também a biblioteca JCL. Abra o Delphi e verifique a existência de várias paletas de componentes.

Validações possíveis com o JEDI VCL

Podemos ter vários tipos de validação através do componente *JvValidator*. Quando você acrescenta a validação já estará escolhendo o seu tipo. Veja na **Tabela 1** os tipos possíveis.

Componente	Função
RequiredFieldValidator	Exige o preenchimento de um campo.
CompareValidator	Comparar o conteúdo de um campo.
RangeValidator	Valida o preenchimento em uma faixa de valores.
RegularExpressionValidator	Valida o preenchimento através de uma Expressão Regular.
CustomValidator	Possibilita a criação de funções próprias para validar.
ControlsCampareValidator	Comparar valores de controles diferentes.

Tabela 1. Tipos de validações possíveis

O mais interessante da criação das regras de validação, é que podemos validar qualquer componente visual, ou seja, que apareça em tempo de execução. E ainda podemos escolher qual propriedade do componente queremos validar, sendo essa uma implementação que não existe nos componentes ASP.NET.

Validações de campo requerido

Vamos iniciar nossa aplicação, colocando a validação mais utilizada: a do campo que deve ser preenchido. Adicione no formulário um *Edit*, um *Button*, um *JvValidator* e um *JvErrorIndicator* (esses dois últimos da paleta *Jv Validators*). No *JvValidator* configure a propriedade *ErrorIndicator* para o *JvErrorIndicator*.

Para adicionar a regra de validação, clique duas vezes sobre o *JvValidator*. Com o editor aberto clique em *New*. Altere a propriedade *ControlToValidate* indicando o *Edit*, a propriedade *ErrorMessage* adicionando o texto "Campo deve ser preenchido" e a propriedade *PropertyToValidate* selecionando *Text*. Adicione o seguinte código no botão:

```
if JvValidators1.Validate then ShowMessage('Validado');
```

O código anterior chama o método de validação do componente, que retorna *True* se todas as regras de validações forem atendidas. Caso alguma regra não seja atendida, o *ErrorIndicator* coloca uma imagem ao lado do campo que não está correto (semelhante ao que é feito em aplicações Windows Forms do .NET). Ao posicionar o mouse em cima da imagem, será exibido um *Hint* contendo a mensagem de texto (**Figura 1**).

Comparando valores

Quando você precisa que um campo seja preenchido com um determinado valor já conhecido, por exemplo, um número inteiro maior que zero, utilizamos uma regra do tipo *CompareValidator*. Clique duas vezes sobre o *JvValidator* e clique na seta ao lado do botão *New*. Serão exibidos todos os tipos de regras que podemos utilizar no componente (**Figura 2**). Escolha *CompareValidator*.

Insira outro *Edit* ao formulário e apague sua propriedade *Text*. Volte à regra de comparação e altere a propriedade *ControlToValidate* para *Edit2*, *ErrorMessage* para "Valor deve ser maior que zero"` e *Operator* para *vcoGreaterThan*. Altere tam-

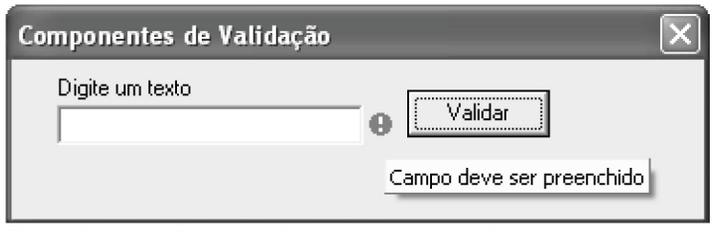


Figura 1. Aplicação validando um campo requerido

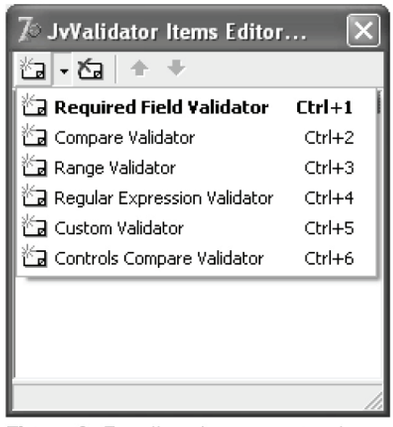


Figura 2. Escolhendo uma regra de comparação

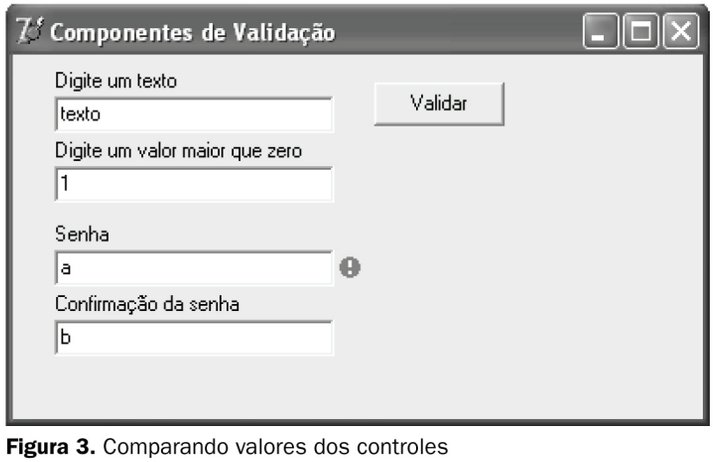


Figura 3. Comparando valores dos controles

bém, *PropertyToValidate* para *Text*, *ValueToCompare* para "0" (zero) e *ValueToCompare.Type* para *Integer*. Execute a aplicação e faça o teste.

Comparando Controles

Também podemos criar uma regra para comparar valores em controles diferentes. Vamos utilizar o exemplo de senha e sua confirmação, portanto adicione mais dois *Edits* ao formulário e apague a propriedade *Text* dos mesmos. Clique duas vezes sobre o *JvValidator* e adicione uma regra do tipo *Controls Compare Validator*. Altere a propriedade *CompareToControl* para *Edit4*, *CompareToProperty* para "Text" e *ControlToValidade* para *Edit3*. Altere também, *ErrorMessage* para "Senha e sua confirmação não conferem", *Operator* para *vcoEqual* e *PropertyToValidate* para *Text*. Veja na **Figura 3** o exemplo em execução.

Faixa de valores

Outra possibilidade é validar o preenchimento de um controle através de uma faixa de valores. Para exemplificar, vamos criar uma validação para um campo que deve ser preenchido com um número inteiro entre 2 e 5. Adicione mais um *Edit* ao formulário e clique duas vezes no *JvValidator* para criar uma nova regra do tipo *Range Validator*. Altere as propriedades *ControlToValidade* para *Edit5* e *ErrorMessage* para "Valor fora da Faixa". Altere também *MaximumValue* para "5", *MaximumValue.Type* para *Integer* e *MinimunValue* para "2". Para finalizar, configure *MinimunValue.Type* para *Integer* e *PropertyToValidade* para *Text*. Rode a aplicação e veja a validação em funcionamento (**Figura 4**).

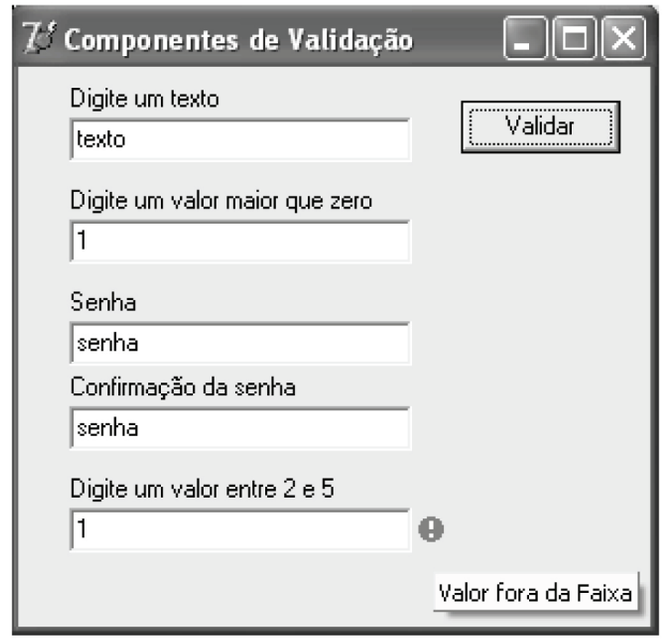


Figura 4. Validando uma faixa de valores com o Range Validator

Utilizando Expressões Regulares

O *JvValidator* permite a utilização de Expressões Regulares em regras de validação. Vamos exemplificar, criando uma regra para validar a entrada de uma letra qualquer, não podendo ser números. Adicione outro *Edit* ao formulário, clique duas vezes no *JvValidator* e adicione uma regra do tipo *Regular Expression Validator*. Configure as propriedades *ControlToValidade* para *Edit6*, *ErrorMessage* para "Digite uma letra de A a Z" e *PropertyToValidade* para *Text*. Em *ValidationExpression* digite: "[a-z]".

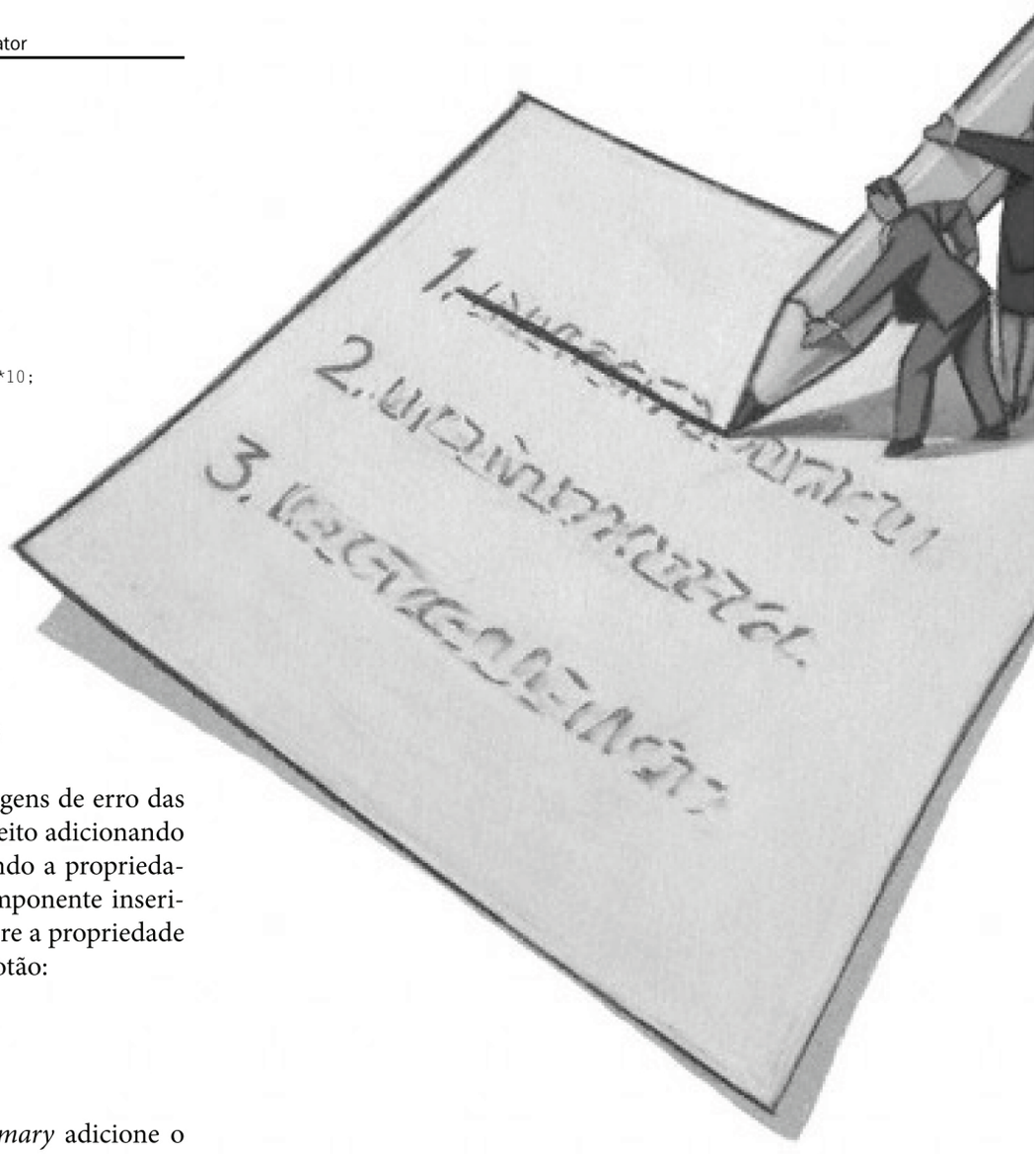
> **Nota:** Para saber mais sobre Expressões Regulares, veja o artigo de Cristiano Caetano na edição 67 da ClubeDelphi.

Customizando validações

Temos a possibilidade de customizar uma validação através de uma função no Delphi. Vamos demonstrar, criando uma validação de CPF. Coloque um *MaskEdit* no formulário, apague sua propriedade *Text* e defina a propriedade *EditMask* para "000.000.000-00;1;". Clique duas vezes e adicione uma regra do tipo *Custom Validator*, alterando as propriedades *ControlToValidate* para *MaskEdit1*, *ErrorMessage* para "CPF Incorreto" e *PropertyToValidade* para *Text*. Para colocarmos o código customizado, acesse o evento *OnValidate* da regra e adicione o código da **Listagem 1**.

Listagem 1. Código do evento OnValidate do Custom Validator

```
var
  n1, n2, n3, n4, n5, n6, n7, n8, n9: integer;
  d1, d2: integer;
  digitado, calculado, cpf: string;
begin
   cpf := VarToStr(ValueToValidate);
   n1 := StrToInt(cpf[1]);
   n2 := StrToInt(cpf[2]);
   n3 := StrToInt(cpf[3]);
   n4 := StrToInt(cpf[5]);
   n5 := StrToInt(cpf[6]);
   n6 := StrToInt(cpf[7]);
   n7 := StrToInt(cpf[9]);
   n8 := StrToInt(cpf[10]);
   n9 := StrToInt(cpf[11]);
   d1 := n9*2+n8*3+n7*4+n6*5+n5*6+n4*7+n3*8+n2*9+n1*10;
   d1 := 11-(d1 mod 11);
   if d1>=10 then d1:=0;
     d2:=d1*2+n9*3+n8*4+n7*5+n6*6+n5*7+n4*8+n3*9+
       n2*10+n1*11;
   d2:=11-(d2 mod 11);
   if d2>=10 then
    d2:=0;
   calculado := IntToStr(d1)+IntToStr(d2);
   digitado := cpf[13]+cpf[14];
   if calculado = digitado then
     Valid := True
   else
     Valid := False;
end;
```



Sumarizando o resultado das validações

Outra opção interessante é exibir as mensagens de erro das validações em um único local. Isso pode ser feito adicionando o *JvValidationSummary* ao formulário e ligando a propriedade *ValidationSummary* do *JvValidator* ao componente inserido. Acrescente ao formulário um *Memo* e altere a propriedade *Name* para "memErros". Altere o código do botão:

```
JvValidationSummary1.Summaries.Clear;
memErros.Lines.Clear;
if JvValidators1.Validate then
  ShowMessage('Validado');
```

No evento *OnChange* do *JvValidationSummary* adicione o seguinte código:

```
memErros.Lines.Text :=
  TJvValidationSummary(Sender).Summaries.Text;
```

Na **Figura 5** temos a aplicação em execução.

> **Dica:** O *JvErrorIndicator* é o responsável por exibir uma imagem ao lado do campo que não foi validado corretamente, inclusive exibindo a mensagem de erro em forma de *Hint*. É possível utilizar outras imagens através de suas propriedades *Images* e *ImageIndex*. Em *BlinkRate*, podemos especificar o tempo da "piscagem" e em *BlinkStyle* definir o tipo de aviso.

O componente JvValidateEdit

O JEDI VCL possui também um componente do tipo *Edit*, que realiza algumas validações bem interessantes, chamado *JvValidateEdit*. Com ele, podemos definir o tipo de entrada de dados como letras ou números, o que fará com que o componente bloqueie a entrada de dados incorretos.

A primeira propriedade que devemos definir é *DisplayFormat* que estabelece o formato de dados que será aceito no momento da digitação. Nos formatos temos vários tipos de dados como: inteiro, real, alfanumérico, binários etc.

Também existe a possibilidade de estabelecer limites de valores, quando utilizados números, podendo definir mudança de cor no texto se o valor digitado está fora de uma faixa. Vamos colocar a mão na massa e criar uma nova aplicação para exemplificar a utilização desse ótimo componente.

Adicione três *JvValidateEdits* no formulário. Para o primeiro, vamos fazer as configurações necessárias para que aceite somente números inteiros e avise se o valor digitado está fora da faixa de 0 a 5. Altere a propriedade *DisplayFormat* para *dfInteger*. Abra as sub-propriedades de *CriticalPoints*. Defina *CheckPoints* para *cpBoth*, *MaxValue* para "5" e *MinValue* para "0". Você também pode alterar as cores através de *ColorAbove* (acima) e *ColorBelow* (abaixo).

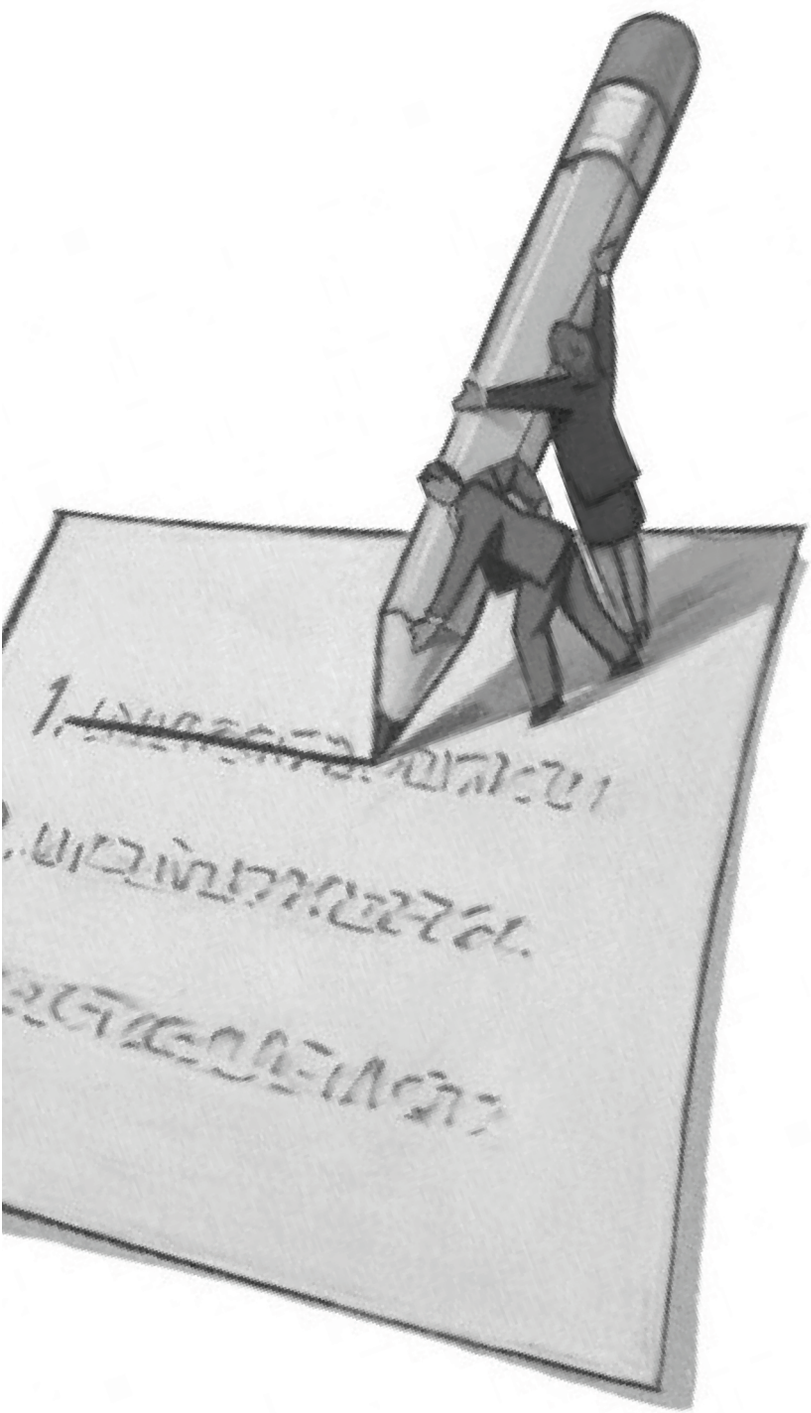
Para o segundo vamos permitir somente letras do alfabeto. Defina a propriedade *DisplayFormat* para *dfAlphabetic*. Para o terceiro também permitiremos a entrada de letras, mas vamos definir que o componente transforme tudo que digitarmos em maiúsculas. Defina a propriedade *DisplayFormat* para *dfAl-*

phabetic e *CharCase* para *ecUpperCase*.

Pronto você já pode testar a aplicação. Perceba que a verificação da faixa de valor só acontece quando o foco sai do componente (**Figura 6**).

Conclusões

Os componentes de validação do JEDI VCL são uma boa alternativa para centralizar as regras de validações de suas aplicações, simplificando em muito a codificação para validar e exibir mensagens de erros. Também é valida a instalação e uso dos componentes JEDI VCL para que essa importante comunidade de desenvolvedores cresça e nos auxilie cada vez mais. ■



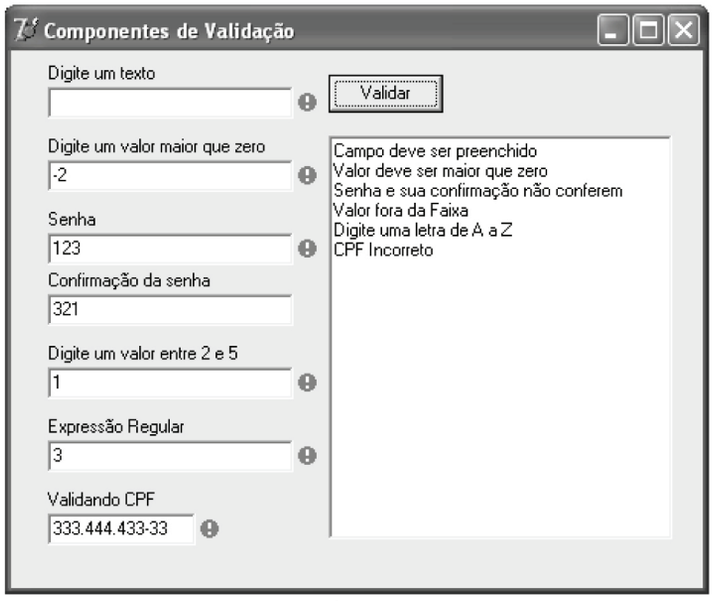


Figura 5. Aplicação exemplo em funcionamento

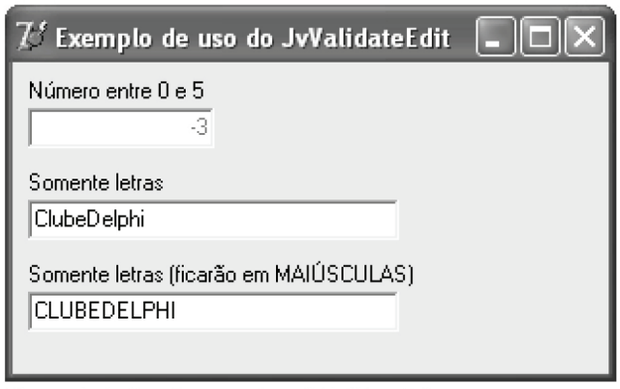


Figura 6. Aplicação usando o JvValidateEdit

Links

Site oficial d o projeto JEDI

www.delphi-jedi.org

Site do JEDI VCL

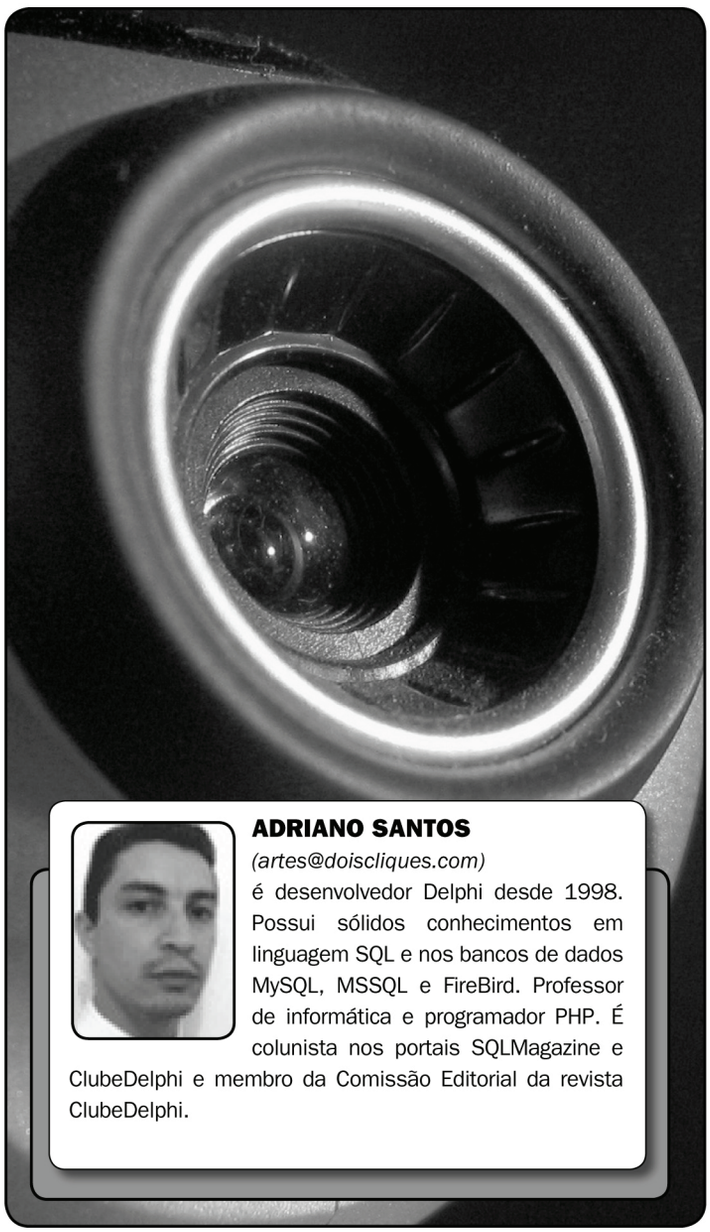
homepages.borland.com/jedi/jvcl

CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como utilizar componentes de validação para Web.

www.devmedia.com.br/articles/viewcomp.asp?comp= 529

Usando Webcam com Delphi



ADRIANO SANTOS
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Possui sólidos conhecimentos em linguagem SQL e nos bancos de dados MySQL, MSSQL e FireBird. Professor de informática e programador PHP. É colunista nos portais SQLMagazine e ClubeDelphi e membro da Comissão Editorial da revista ClubeDelphi.

A segurança tem sido uma das maiores preocupações em empresas e edifícios comerciais. Por isso, é cada vez mais freqüente o uso de câmeras de vigilância, cadastros mais complexos, uso de crachás e uma série de outras medidas preventivas para evitar problemas.

Hoje em dia, é comum, principalmente na recepção de edifícios comerciais, o cadastro obrigatório do visitante informando dados como: nome, endereço, idade, documento de identidade, data, horário, andar, sala e contato visitado. Há também empresas que fazem o "escaneamento" do documento de identidade.

Com o preço de hardware cada vez mais acessível, uma boa opção pra quem quer incrementar ainda mais a segurança é usar uma Webcam para tirar uma foto do visitante e armazená-la no banco de dados, ou seja, com um valor médio de R$ 90,00 é possível adquirir uma boa câmera e com isso desenvolver um sistema de cadastro bastante completo.

Neste artigo desenvolveremos um sistema que fará uso de uma Webcam para salvar a foto do visitante.

Entendendo o projeto

Nossa aplicação receberá o nome de "Recepcao", onde teremos uma tela simples, com todas as informações do visitante do edifício ou empresa. Serão registradas informações como: Nome, Endereço, Cidade, Estado, Telefone e ainda sua foto.

Seu cadastro ficará registrado e consequentemente sua visita (tabela VISITAS), dessa forma, toda vez em que essa pessoa retornar à empresa será registrada um novo evento.

> **Dica:** Na tabela VISITANTES, guardaremos apenas o endereço da foto. O nome do arquivo será o RG sem pontos. Ex: 289604368.jpg. O caminho completo ficaria algo como: C:\Recepcao\Fotos\289604368.jpg ou o que melhor convir para o desenvolvedor.

Não entraremos em detalhes sobre a estrutura das tabelas, pois nosso foco é a utilização da WebCam com Delphi. Citaremos apenas os principais campos das tabelas que guardarão as informações do visitante.

Repetiremos os campos EMPRESA e CONTATO na tabela VISITAS, isso porque se for um prédio comercial podem acontecer casos em que a pessoa necessitará visitar mais de uma empresa no mesmo edifício, como ocorrem com motoboys, office-boys, vendedores e empresas de cobrança.

Esses campos trarão como sugestão os respectivos valores registrados na tabela VISITANTES nos campos de mesmo nome. Para atualizarmos os campos identificadores (ID) de ambas tabelas usaremos *Triggers* e *Generators* automatizando essas tarefas.

O componente TCamera

A foto será tirada por meio de uma Webcam com o auxílio de um componente que instalaremos no Delphi. Tomei o cuidado de escolher um que fosse *freeware*, pois o procedimento é bastante simples. O *TCamera* pode ser baixado com os fontes do artigo. Para instalar basta abrir o arquivo *TWebCam.dpk*, compilar e instalar.

A principio, o componente não tem restrição de versão do Delphi, tendo sido testado no Delphi 7 e Delphi 2006.

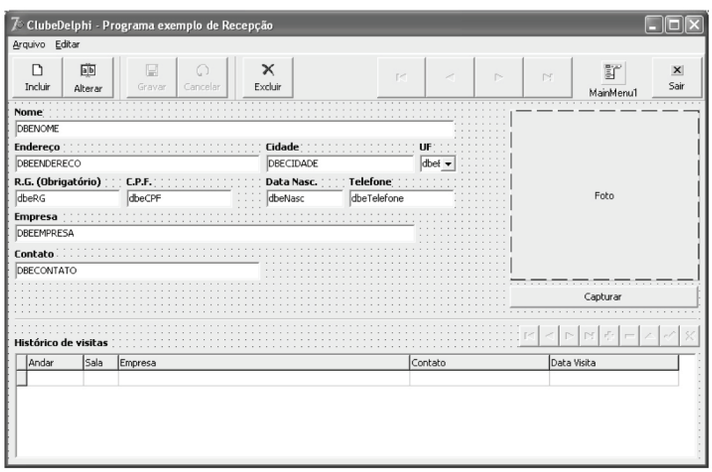


Figura 1. Tela principal do sistema Recepção

Desenhando o cadastro

O exemplo trata de uma janela mestre/detalhe, onde vemos os dados do visitante e logo abaixo, suas visitas. O exemplo, como mencionado, é simples e dispensa maiores detalhes, por isso vamos nos focar no desenvolvimento de captura da foto. Nossa janela deverá ser semelhante a **Figura 1**.

Note que, no canto direito, temos um *Panel* e dentro dele um *Image*. É nesse local que visualizaremos a foto dos visitantes. Essa foto será atualizada no evento *OnChange* do *DataSource* que será ligado à tabela VISITANTES.

A busca da foto será feita usando o diretório do executável concatenado com o número do RG do visitante. A extensão será JPG para economizar espaço em disco.

Criando o Data Module

Para acesso a base de dados, vamos utilizar dbExpres, com o que eu chamo de "Quarteto Fantástico": *SQLConnection, SQLDataSet, DataSetProvider e ClientDataSet*. Crie um Data Module, salve-o como "fDmPrincipal.pas" e digite em *Name* "DmPrincipal".

Adicione um *SQLConnection* e configure-o de maneira a conectar na base de dados. Mude a propriedade *Name* para "sqlConexao". Insira alguns componentes, baseando-se na **Figura 2**.

Para auxiliar em pesquisas, *queries* ou mesmo executar algum tipo de comando SQL, criamos um trio de componentes que possa executar tais tarefas (com o nome de *Auxiliar*). Adicione na propriedade *CommandText* dos *SQLDataSets* o seguinte código SQL:

```
sdsVisitantes
SELECT * FROM VISITANTES
sdsVisitas
SELECT * FROM VISITANTES WHERE
  ID_VISITANTE =:ID_VISITANTE
```

A programação do Data Module é bem simples, onde começaremos trabalhando com o evento OnAfterScroll do ClientDataSet de visitantes (cdsVisitantes). Vamos usar um mecanismo bem simples para controlar os detalhes dessa tabela, ou seja, as visitas realizadas por nosso "visitante". Optei por fazer o controle manualmente, mas existem outras técnicas, portanto use a que você mais gostar. Digite o código da **Listagem 1**.

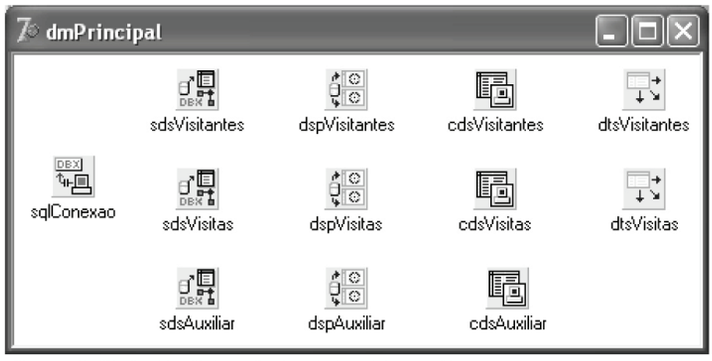


Figura 2. Data Module principal do sistema Recepção

Listagem 1. Evento OnAfterScroll do cdsVisitantes.

```
procedure TdmPrincipal.cdsVisitantesAfterScroll(
  DataSet: TDataSet);
begin
  with cdsVisitas do
  begin
    Close;
    Params.ParamByName(
      'ID_VISITANTE').AsInteger :=
      DataSet.FieldByName('ID').AsInteger;
    Open;
  end;
end;
```

Seguindo a mesma linha de simplicidade, o código para atualização do Image é bem compacto. Basta pegar o endereço completo da imagem no campo FOTO da tabela VISITANTES e chamar o método LoadFromFile do Image presente no formulário principal. Veja o código na **Listagem 2**.

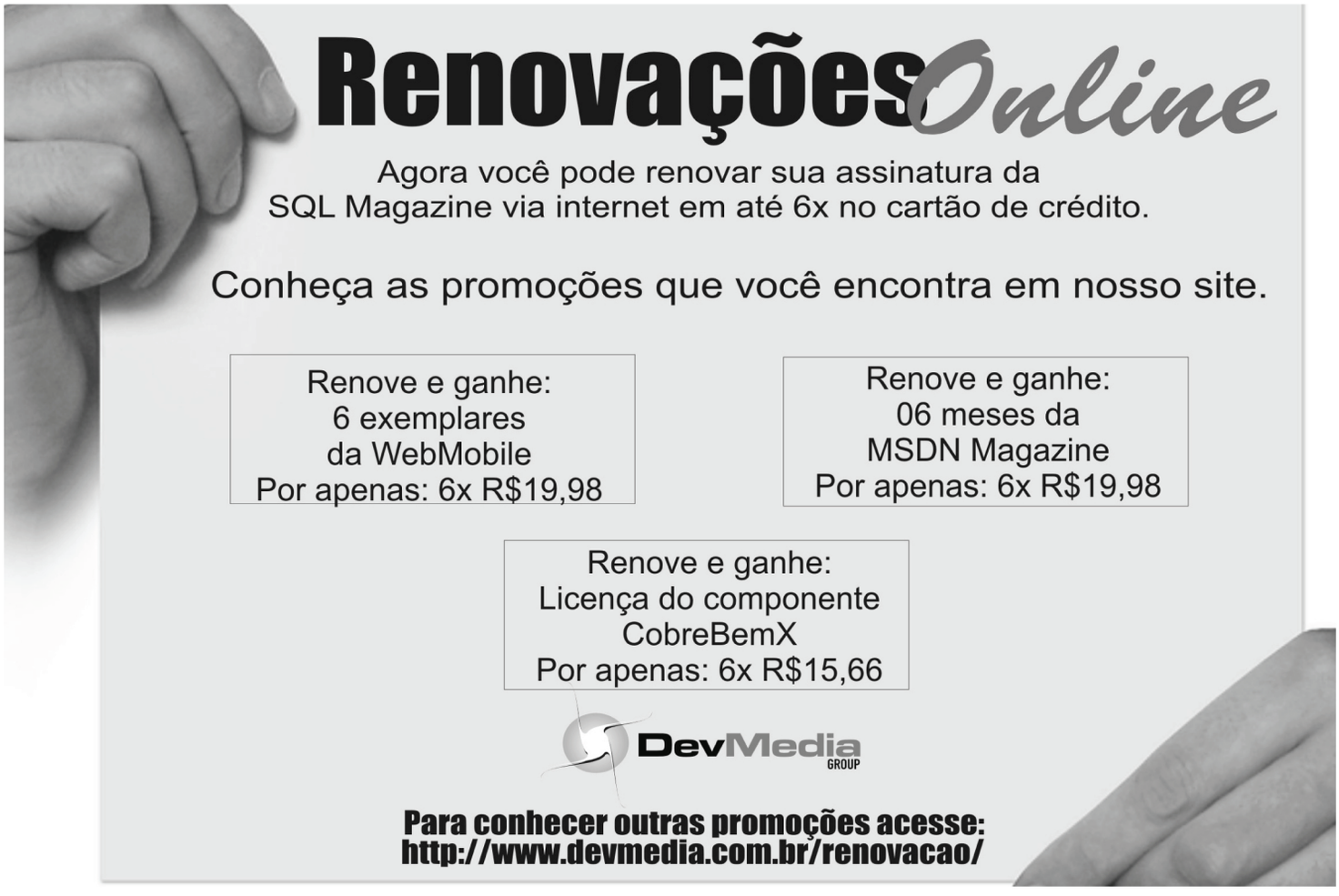
Listagem 2. Evento OnChange do dtsVisitantes usado para atualizar a foto do visitante no Imagem

```
procedure TdmPrincipal.dtsVisitantesDataChange(
  Sender: TObject; Field: TField);
var
  strCaminho: string;
  strRG: string;
begin
  if not cdsVisitantes.IsEmpty then
    if FileExists(cdsVisitantes.
      FieldByName('FOTO').AsString) then
      fPrincipal.imgFoto.Picture.LoadFromFile(
        cdsVisitantes.FieldByName(
        'FOTO').AsString)
    else
      fPrincipal.imgFoto.Picture := nil;
end;
```

Figura 3. Design da janela fCaptura

> **Nota:** Para simplificar o exemplo, acessei o formulário diretamente a partir do DM, mas lembre-se que essa prática não é recomendada em uma aplicação real.

Pra finalizar a programação no DM, basta inserir o código que trará como sugestão os valores dos campos EMPRESA e CONTATO da tabela visitantes (**Listagem 3**).



Listagem 3. Preenchendo o valor dos campos EMPRESA e CONTATO

```
procedure TdmPrincipal.cdsVisitasAfterInsert(
  DataSet: TDataSet);
begin
  cdsVisitas.FieldByName('EMPRESA').AsString :=
    cdsVisitantes.FieldByName('EMPRESA').AsString;
  cdsVisitas.FieldByName('CONTATO').AsString :=
    cdsVisitantes.FieldByName('CONTATO').AsString;
end;
```

Nesse ponto finalizamos o DM, agora vamos voltar à tela principal. Não vou entrar em detalhes quanto ao desenvolvimento do formulário principal, pois basta incluir rotinas simples para Inclusão, Alteração, Exclusão e Gravação dos dados em ambas as tabelas (o código completo está para download). Vamos nos focar no TCamera.

Capturando a imagem

Crie um novo formulário, dando o nome de "fCaptura". Abaixo do *Panel*, no formulário principal, temos um botão chamado *btnCaptura*. Ele será responsável por ativar a Webcam e capturar a foto do usuário.

Após a criação desse formulário, inclua um TCamera. Ele se parece bastante com um Panel. Alinhe-o conforme a **Figura 3** e insira dois novos botões.

Para o botão *Confirmar*, em *Name* digite "btnConfirmar" e escolha em *ModalResult* a opção *mrOK*. Para o *Cancelar* digite "btnCancelar" para o nome e escolha *mrCancel* em *ModalResult*.

Nessa janela, faremos apenas a ativação e desativação da Webcam através do *TCamera*. O *TCamera* trabalha com um ponteiro, configurado para um *array de bytes*. Inclua na declaração na seção interface da *unit* o seguinte código:

```
type
  TTabImage = Array[0..230399] of byte;
```

Após, codificamos o evento *OnShow* do formulário para ativar o componente (**Listagem 4**).

Listagem 4. Ativando o componente

```
procedure TfCaptura.FormShow(Sender: TObject);
var
  P: ^TTabImage;
begin
  try
    camCamera.Actif := True;
    New(P);
    fCaptura.Caption := fCaptura.Caption +
      camCamera.FichierImage;
  except
    MessageDlg('Ocorreu um erro ao tentar ativar '+
      'a Webcam.' + #13 + 'Certifique-se de que a '+
      'câmera esteja conectada e corretamente '+
      'configurada.', mtError, [mbOk], 0);
  end;
end;
```

A seguir, desativamos o componente no evento *OnDestroy*:

```
procedure TfCaptura.FormDestroy(Sender: TObject);
begin
  camCamera.Actif := False;
end;
```

Nesse ponto, nosso exemplo está quase concluído, faltando apenas encerrarmos com a captura e gravação da foto no disco. Para isso, volte ao formulário principal para codificar o botão Capturar.

Uma explicação breve sobre o componente: quando ativado o mesmo permanecerá "filmando" as imagens até que seja desativado. A foto consiste apenas em fazer uma chamada ao método CaptureImageDisque. Esse método "congela" a imagem e a salva no disco.

O nome da imagem bem como sua pasta de destino são determinadas pela propriedade FichierImage. Vale lembrar que o componente trabalha apenas com imagens no formato BMP, o que não é muito interessante, pois geram arquivos grandes.

Para resolver essa situação, que não chega a ser um problema, inclui uma rotina que faz a conversão de arquivos BMP para JPEG.

O nome da imagem, será o número do RG do visitante, sem máscara, e será salva na pasta do executável. O diretório e nome de foto serão gravados no campo FOTO da tabela VISITANTES. Para finalizar, digite o código da **Listagem 5** no evento OnClick do btnCaptura.

Veja o sistema em funcionamento nas **Figuras 4** e **5**.

Conclusão

A preocupação com segurança nos leva a criar cada vez mais artifícios para controlar a entrada e saída de pessoas em empresas e edifícios comerciais. Com este artigo aprendemos a incluir em nosso sistema uma forma de fotografar o visitante e ter um registro mais completo. ■

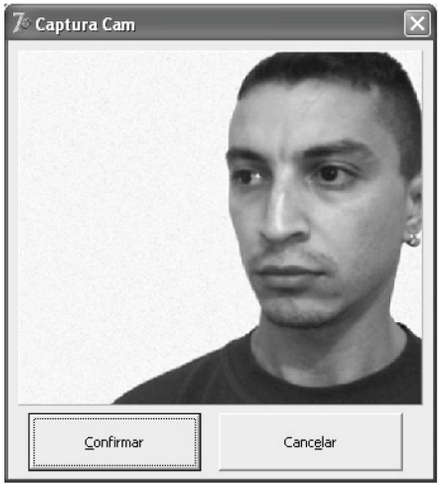


Figura 4. Janela de captura da foto

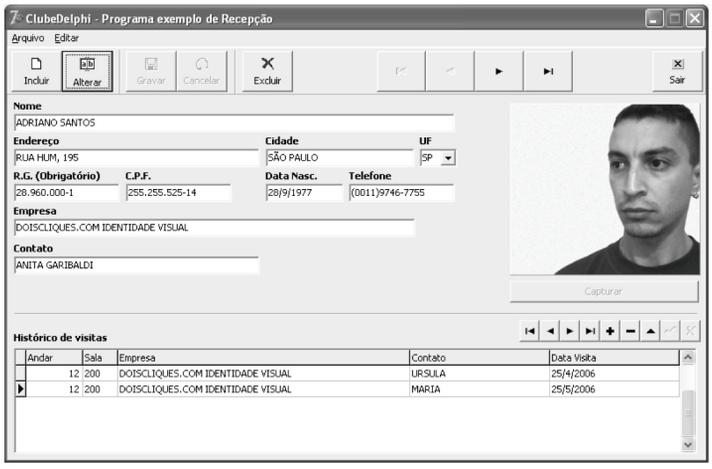
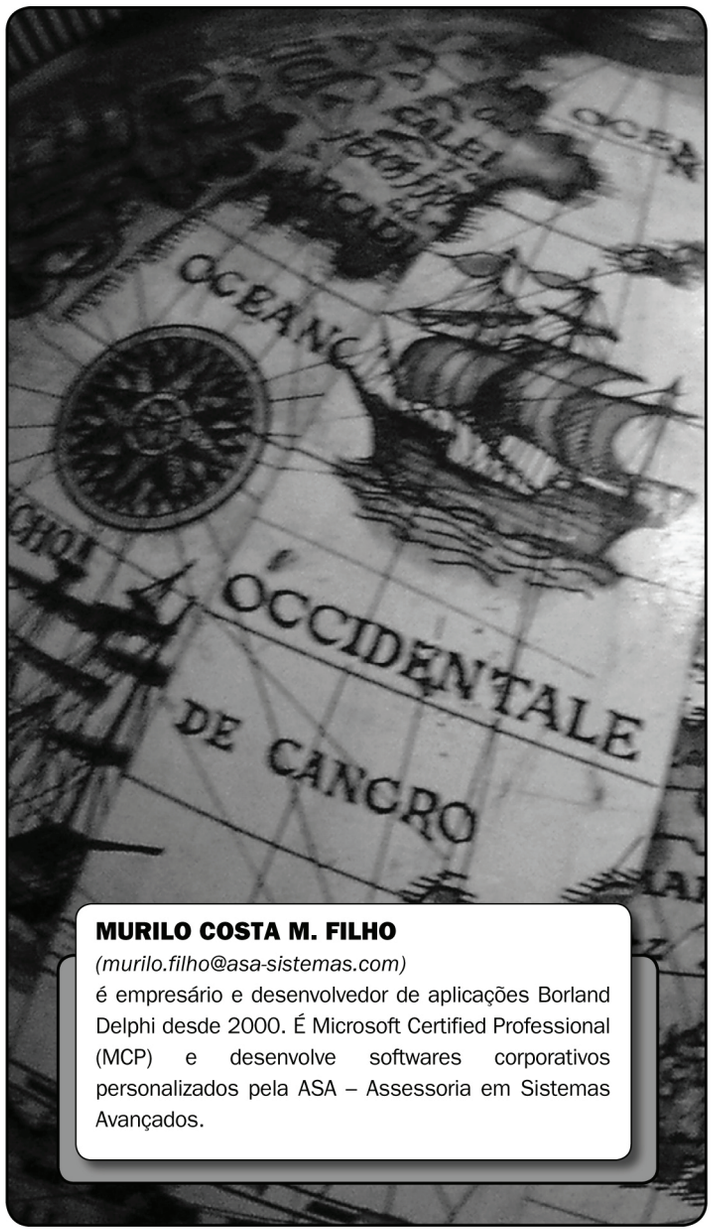


Figura 5. Janela principal já com o cadastro completo

Listagem 5. Evento onClick do btnCaptura, responsável pela captura e gravação da foto

```
procedure TfPrincipal.btnCapturaClick(Sender: TObject);
var
  DirNomeFoto: string;
  NomeFoto: string;
  procedure ConverteParaJpeg(ACaminhoFoto: string);
  var
    cjBmp: TBitmap;
    cjJpg: TJpegImage;
    strNomeSemExtensao: string;
    AFoto: TImage;
  begin
    AFoto:= TImage.Create(fPrincipal);
    AFoto.Parent := fPrincipal;
    AFoto.Visible := False;
    AFoto.Picture.Bitmap.LoadFromFile(
      ACaminhoFoto + '.bmp');

    cjJpg := TJPegImage.Create;
    cjBmp := TBitmap.Create;

    cjBmp.Assign(AFoto.Picture.Bitmap);
    cjJpg.Assign(cjBMP);

    cjJpg.SaveToFile(ACaminhoFoto + '.jpg');
    DeleteFile(ACaminhoFoto + '.bmp');
    cjJpg.Free;
    cjBmp.Free;
    AFoto.Free;
  end;
begin
  fCaptura := TfCaptura.Create(Self);
  try
    with fCaptura, camCamera, dmPrincipal,
      cdsVisitantes do
    begin
      DirNomeFoto := ExtractFilePath(
        Application.ExeName) + LimpaString(FieldByName(
        'RG').AsString) + '.bmp';

      NomeFoto := ExtractFilePath(DirNomeFoto) +
        Copy(ExtractFileName(DirNomeFoto),1, Length(
        ExtractFileName(DirNomeFoto))-4);

      FichierImage := ExtractFileName(DirNomeFoto);
      if fCaptura.ShowModal = mrOk then
      begin
        CaptureImageDisque;
        ConverteParaJpeg(NomeFoto);
        FieldByName('FOTO').AsString :=
          NomeFoto + '.jpg';
      end;
    end;
  finally
    FreeAndNil(fCaptura);
  end;
end;
```

Configurando o Windows 2003 Server para aplicações DataSnap



MURILO COSTA M. FILHO
(murilo.filho@asa-sistemas.com)
é empresário e desenvolvedor de aplicações Borland Delphi desde 2000. É Microsoft Certified Professional (MCP) e desenvolve softwares corporativos personalizados pela ASA – Assessoria em Sistemas Avançados.

Na edição 70 vimos no artigo de Guinther Pauli como criar aplicações distribuídas utilizando a tecnologia DataSnap e o protocolo SOAP. Para que seja possível acessar a DLL do projeto servidor, é necessário ativar o serviço de conteúdo dinâmico no servidor IIS (*Internet Information Services*), caso o usuário esteja utilizando Windows Server 2003 com o IIS na versão 6. Neste artigo, veremos como configurar o servidor Windows 2003 para suportar aplicações Delphi com DataSnap.

Verificando o erro

Por padrão, a instalação do IIS 6.0 no Windows Server 2003 somente aceita requisições para conteúdos estáticos (HTML). Todo o conteúdo dinâmico (CGI, ISAPI, ASP etc.) é bloqueado e necessita ser habilitado manualmente. Nesse caso, se o usuário tentar acessar uma página ASP ou uma DLL ISAPI ou CGI, o IIS exibirá uma mensagem de erro HTTP 404 - File Not Found. O Delphi retornará a mensagem de erro: Not Found (404), seguida do caminho da DLL, caso o usuário tente acessar o servidor DataSnap através do aplicativo cliente (**Figura 1**).

Solucionando o problema

Inicie o *IIS Manager*, através do menu *Iniciar>Painel de Controle>Ferramentas Administrativas*, ou então digite "inetmgr" no *Executar*. No gerenciador, clique na pasta *Web Service Extensions*, na árvore de navegação. No painel da direita, nota-se que todos os serviços de conteúdo dinâmico estão com a propriedade *Status* marcada como *Prohibited* (**Figura 2**), significando que o servidor IIS está proibido de aceitar chamadas a essas extensões.

Figura 1. Mensagem de erro do projeto cliente

Nota: É possível solucionar o problema clicando com o botão direito em *All Unknown CGI Extensions* e clicar em *Allow*. No entanto, não é recomendável permitir a execução de todas as extensões CGI por questões de segurança. Nesse caso, habilitaremos o acesso somente à aplicação servidora do exemplo da edição 70.

No painel da esquerda, em *Tasks*, clique em *Add a new Web Service Extension*. Na janela *New Web Service Extension*, insira o nome da extensão em *Extension Name*. Em seguida clique no botão *Add* e aponte o caminho para a DLL do servidor DataSnap. Marque a caixa *Set Extension status to Allowed*. A janela deverá estar preenchida como na **Figura 3**. Faça o teste novamente no navegador e veja que agora a DLL do servidor DataSnap foi acessada normalmente.

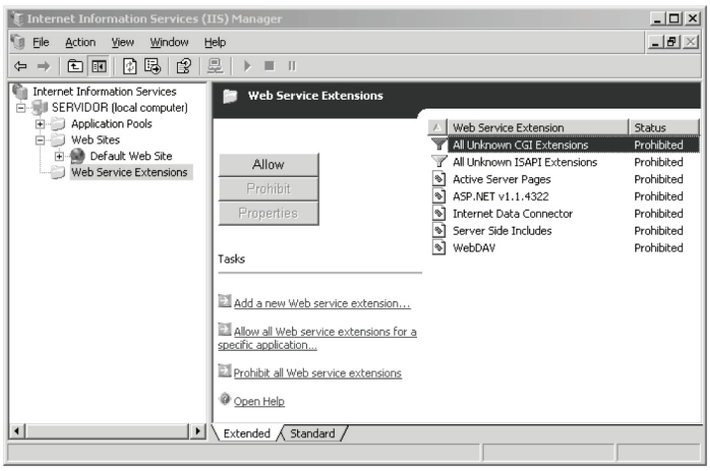


Figura 2. Os serviços de conteúdo dinâmico encontram-se bloqueados

Passos adicionais para configuração de aplicações CGI

No *IIS Manager*, clique com o botão direito em *Default Web Site* e clique em *Properties*. Na janela *Default Web Site Properties*, clique na aba *Home Directory* e na caixa de listagem *Execute Permissions* selecione *Scripts and Executables* (**Figura 4**).

Adicionalmente, é necessário que o grupo de usuários tenha as seguintes permissões na pasta *wwwroot*. Clique com o botão direito na pasta *wwwroot*, que se encontra no diretório *Inetpub*, e clique em *Properties*. Na aba *Security*, clique em *Users* e marque as caixas *Read & Execute*, *List Folder Contents* e *Read* (**Figura 5**).

Pronto, agora o servidor IIS 6.0 está configurado correta-

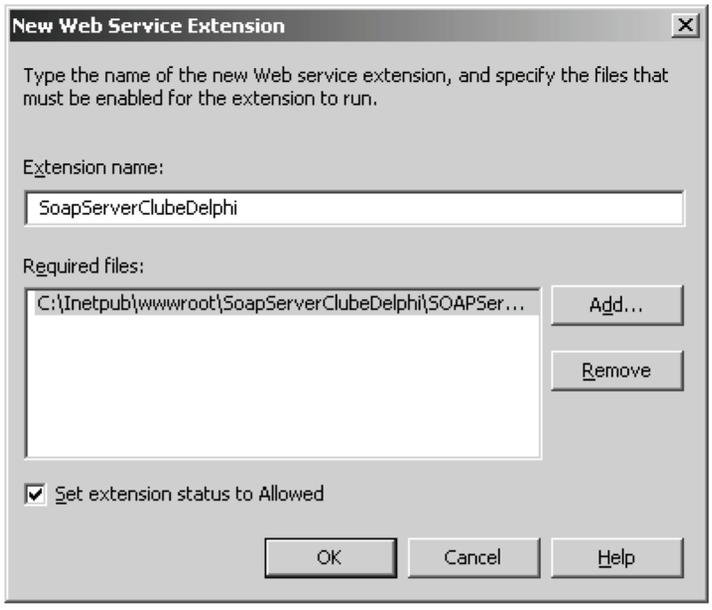


Figura 3. Adicionando uma nova extensão ao servidor IIS

mente para o uso de aplicações CGI e para os testes de suas aplicações DataSnap com o protocolo SOAP. Lembrando que para aplicações adicionais, devemos adicionar as extensões correspondentes como visto anteriormente.

Conclusão

O Windows 2003 Server é sem dúvida a melhor versão do Windows para rodar aplicações Web servidoras, incluindo as construídas com DataSnap no Delphi. Vimos neste artigo os cuidados e passos necessários para que sua aplicação possa funcionar corretamente no Windows 2003. ■

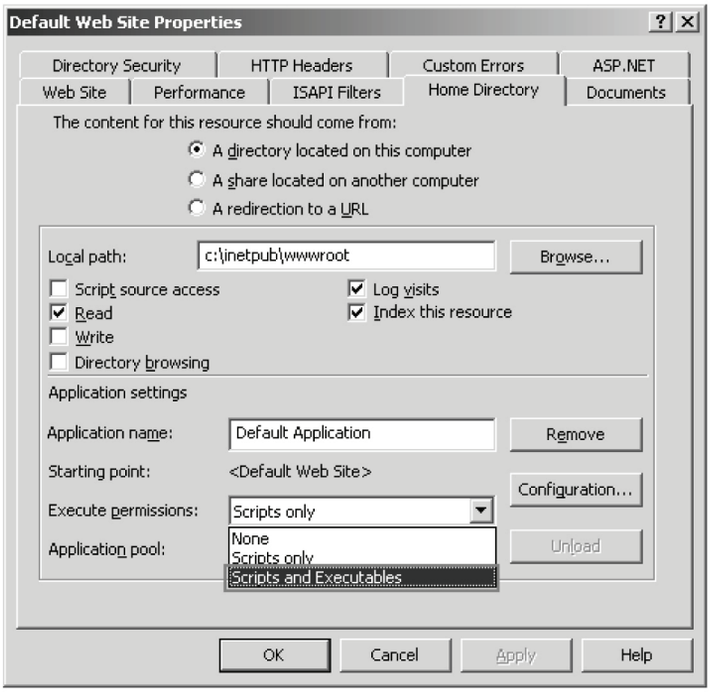


Figura 4. Configurando as permissões de execução

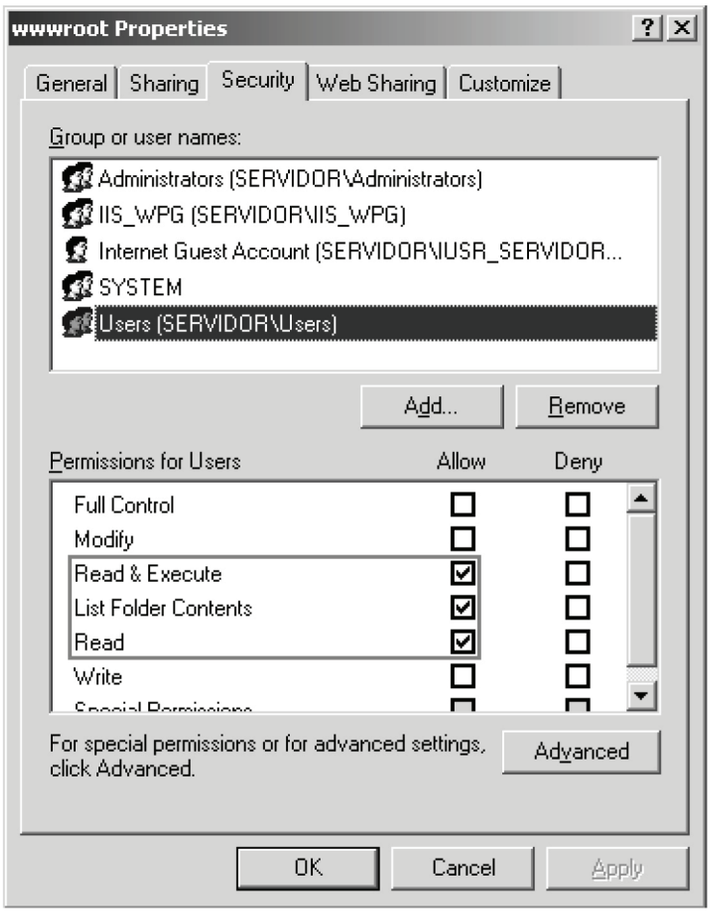


Figura 5. As permissões para o diretório wwwroot

Complete sua coleção!

Feita para Desenvolvedores de Software

SQL magazine

Escolha 03 edições e pague **R$ 6,90** em cada revista.

20

- Mapeamento Objeto Relacional. Aprenda a construir um mapeador O/R em .NET.

- Warehouse Builder. Construa uma data warehouse com a ferramenta ETL da Oracle-Partel

- UML. Aprenda passo a passo como construir um diagrama de sequências.

- Otimização. Influenciando o otimizador de consultas.

21

- Normalização. Aprenda a organizar melhor o esquema de seu banco de dados

- PostreSQL. Saiba o que são rules e como utilizá-las.

- SQL Server 2000. Entenda os modelos de recovery.

- InterBase 7.5. Quando e porque migrar.

- Particionamento do Oracle. Perigos e armadilhas Parte II.

22

- Backup. Saiba como proteger seus dados no Oracle e no SQL Server.

- UML. Conheça algumas opções de ferramentas livres - Parte II.

- Homologação de dados. Saiba como avaliar a qualidade de seu modelo de dados.

- Especial. Acessando servidores Oracle a partir do SQL Server 2000.

23

- Proteja-se. Saiba como funciona o SQL Injection e aprenda a lidar com ele no SQL Server.

- Administrando o Oracle. Conheça o Oracle Enterprise Manager.

- Backup. Saiba como efetuar o restore de Bds no SQL Server.

- Modelagem. Utilizando padrões na elaboração do projeto de banco de dados.

24

- Java + BD + Web. Trabalhe de forma integrada com: MySQL, Tomcat e Java com NetBeans.

- Oracle 10g. Aumente a disponibilidade de seu SGBD com o automatic Storage Management.

- PostgreSQL. Veja na prática como efetuar replicação entre servidores PostgreSQL.

25

- Clustering no SQL Server. Saiba para que serve e como funciona essa tecnologia de redundância.

- IBExpert. Configure e administre base de dados no Firebird e interbase.

- Caché. trabalhe com objetos diretamente no banco de dados.

- Oracle Discoverer. A prenda passo a passo a trabalhar com a ferramenta de análise de dados da Oracle.

26

- Mineração de dados. Entenda como ocorre a descoberta de informações em base de dados.

- IBExpert. Administrando o InterBase/Firebird. Conheça algumas características avançadas.

- UML. Ferramenta de modelagem free com engenharia reversa.

- MySQL. Transforme-se em um especialista na otimização de consultas - Parte II.

27

- MySQL. Veja como rodar um banco de dados MySQL em várias máquinas ao mesmo tempo!

- ER/Studio. Conheça esta excelente ferramenta para modelagem de dados.

- SQL. Descobrindo associações entre pares de linhas.

- SQL Server. Veja como enviar e-mails anexos no SQL Server sem o uso do SQL Mail.

28

- Dúvidas frequentes em bancos de dados.

- Desafio SQL. Construindo um modelo para um jogo de futebol on-line.

- Index Tuning Wizard. Como produzir tunings automáticos.

- Erwin conheça esta ferramenta de modelagem E/R.

29

- Oracle FLASHBACK. Recupere informações para um determinado ponto do tempo.

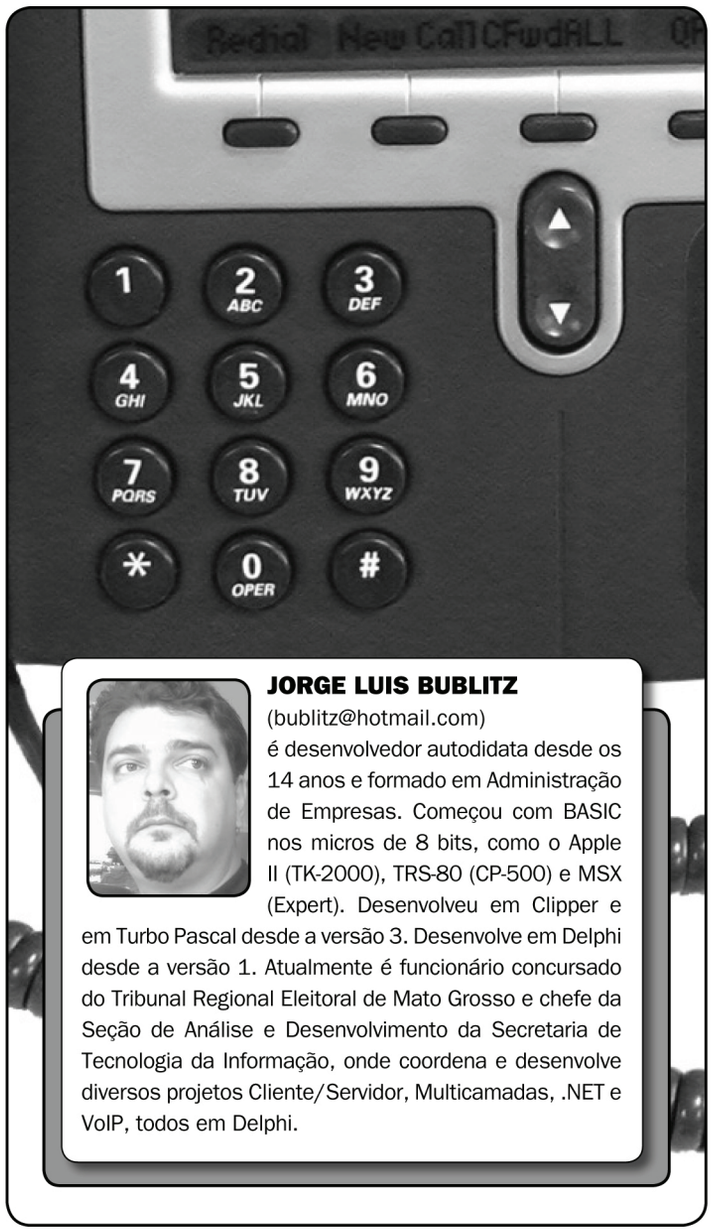
- SQL Server. Descubra se você esta trabalhando corretamente com o SQL Server.

- InterBase / Firebird. Trabalhando com stored procedures e triggers.

www.devmedia.com.br/msdn/anteriores

ou pela nossa central de atendimento: (21)2220-5375

VoIP com Delphi - Desenvolva o seu próprio "Skype"



JORGE LUIS BUBLITZ
(bublitz@hotmail.com)
é desenvolvedor autodidata desde os 14 anos e formado em Administração de Empresas. Começou com BASIC nos micros de 8 bits, como o Apple II (TK-2000), TRS-80 (CP-500) e MSX (Expert). Desenvolveu em Clipper e em Turbo Pascal desde a versão 3. Desenvolve em Delphi desde a versão 1. Atualmente é funcionário concursado do Tribunal Regional Eleitoral de Mato Grosso e chefe da Seção de Análise e Desenvolvimento da Secretaria de Tecnologia da Informação, onde coordena e desenvolve diversos projetos Cliente/Servidor, Multicamadas, .NET e VoIP, todos em Delphi.

A onda do momento chama-se VoIP, sigla para *Voice Over IP*, ou voz sobre IP. Vários provedores e portais estão disponibilizando esse serviço, de graça ou não. O já consagrado software *Skype* está entre os mais baixados no mundo inteiro. Neste artigo veremos como desenvolver uma aplicação VoIP simples, para comunicação entre dois computadores conectados em rede, conhecida como comunicação P2P.

O que é VoIP?

VoIP é uma tecnologia que permite a digitalização e codificação da voz e o encapsulamento desse dado em pacotes IP, para transmissão em uma rede que utilize o protocolo TCP/IP. Voz sobre IP é um conceito relativamente simples, onde basta transformar a voz em um fluxo de bits, que é encapsulado em pacotes UDP (*User Datagram Protocol*, algo como Protocolo de pacote de dados do usuário), que por sua vez são encapsulados em pacotes TCP (**Figura 1**).

Utiliza-se para transmissão o protocolo chamado RTP (*Real-Time Transport Protocol*, ou protocolo de transmissão em tempo real). O RTP utiliza o serviço de transporte UDP para transmitir os pacotes. O RTP é necessário porque na Telefonia IP uma taxa de transmissão constante é fundamental, enquanto que a perda de pacotes pode ser desprezada. Os pacotes IP assim obtidos são transportados pela rede, como qualquer pacote IP, sem qualquer distinção.

O "CD-VoIP", aplicação construída como exemplo deste artigo, é um software que possibilita a conversa via IP entre dois computadores que possuem recursos de áudio (placa de áudio, microfone e caixas de som), conectados em rede e não é necessária a adoção de nenhum hardware especial na rede (como vemos a simplicidade na **Figura 1**). Ele apresenta uma possibilidade de uso dessa tecnologia em substituição ao sistema telefônico convencional, aproveitando uma rede de computadores já existente.

Para realizar uma chamada são necessários protocolos para

executar algumas tarefas, como a localização do usuário, notificação de chamada, início e finalização de transmissão de voz e desconexão, feito através de protocolos de controle.

O Padrão H.323

Hoje a arquitetura mais difundida no mercado para transmitir voz sobre uma rede TCP/IP é o padrão H.323, proposto pela *International Telecom Union* (ITU-T). Esse padrão é uma pilha de protocolos que está focado na conexão e controle da chamada, que são separados da transmissão de conteúdo (voz) entre os computadores (**Figura 2**).

"*O padrão H.323 é completamente independente dos aspectos relacionados à rede. Dessa forma, podem ser utilizadas quaisquer tecnologias de enlace, podendo-se escolher livremente entre as que dominam o mercado atual como Ethernet, Fast Ethernet (FDDI) ou Token Ring. Também não há restrições quanto à topologia da rede, que pode consistir tanto de uma única ligação ponto a ponto, ou de um único segmento de rede, ou ainda serem complexas, incorporando vários segmentos de redes interconectados.*

O H.323 especifica o uso de áudio, vídeo e dados em comunicações multimídia, sendo que apenas o suporte à mídia de áudio é obrigatório. Mesmo sendo somente o áudio obrigatório, cada mídia (áudio, vídeo e/ou dados), quando utilizada, deve seguir as especificações do padrão. Pode-se ter uma variedade de formas de comunicação, envolvendo áudio apenas (telefonia IP), áudio e vídeo (videoconferência), áudio e dados e, por fim, áudio, vídeo e dados" (Wikipedia).

O CD-VoIP tem o seu funcionamento demonstrado na **Figura 3**, onde estabelecemos uma comunicação ponto a ponto, com a sua utilização sendo bastante simples: basta o usuário digitar o endereço IP ou o nome da máquina na rede para estabelecer a ligação.

A aplicação roda o protocolo H.323 sobre TCP, a fim de estabelecer um canal de transmissão e recepção por meio da rede IP. Quando a ligação é atendida, é então estabelecido um fluxo RTP sobre UDP entre o computador de origem e o de destino.

O Blitz VoIP

Ao iniciar o desenvolvimento da aplicação, busquei na Internet informações sobre produtos e componentes de VoIP que fossem *freeware* e/ou *open-source*. Encontrei vários exemplos, nas mais diversas linguagens de desenvolvimento, porém em todos havia um problema: o *codec* de áudio.

Em todos os produtos era necessária a instalação de um codificador-decodificador (*codec*) de áudio, sendo essa forma de aplicação inviável, pois torna-se dependente de aplicação. O que desejava era uma aplicação de simples instalação e uso, do tipo "copie e use", sem a necessidade de instalação de outros softwares.

Ainda na Internet, pesquisando no site MSDN da Microsoft, encontrei a documentação sobre a TAPI, *Telephony Application Program Interface*. O padrão H.323 faz parte da TAPI 3.x, presente nas versões 2000, XP e 2003 do Windows. Com isso não foi necessário adotar nenhum tipo de tecnologia (*codec*) além da já presente no sistema operacional. Contudo o programa não funcionará em Windows 95/98 e NT.

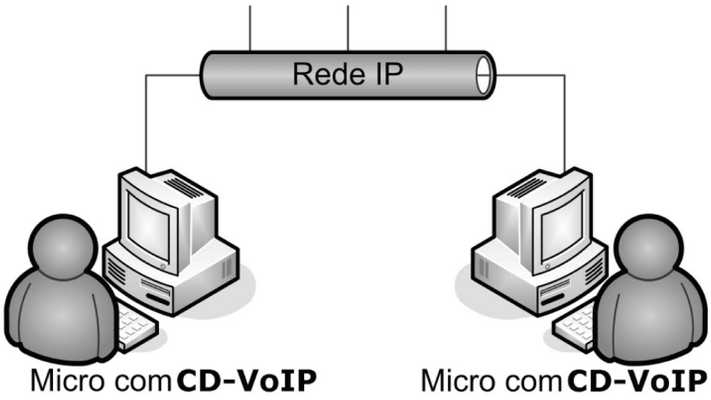


Figura 1. Encapsulação do fluxo de dados de voz em pacotes IP

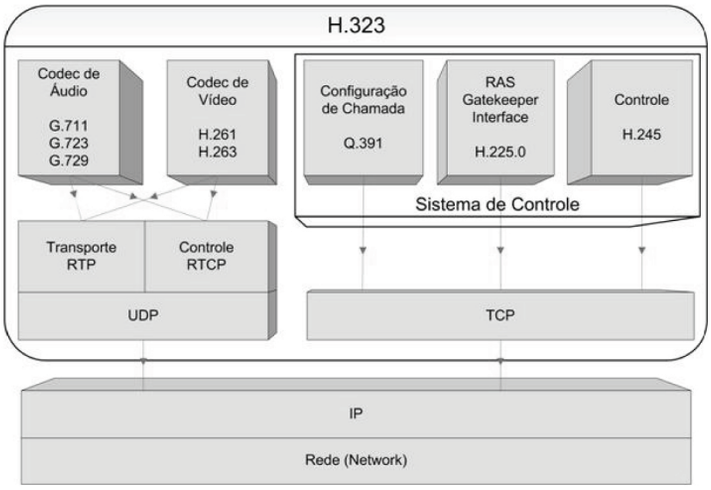


Figura 2. O Padrão H.323

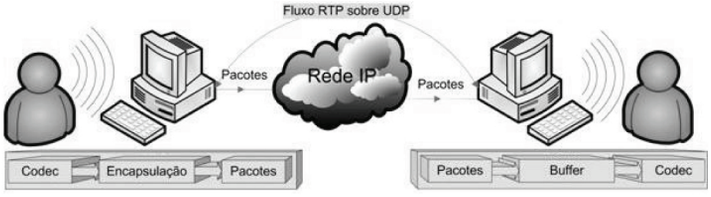


Figura 3. Funcionamento do software CD-Voip

No site da MSDN há diversos exemplos em *Visual C++* de como utilizar a TAPI e o protocolo H.323. Mas não há uma aplicação completa e mesmo com todos os exemplos juntos, foi necessário buscar mais informações em outros sites na Internet (veja na seção *Links*). Assim, juntei tudo isso num único componente: o *Blitz VoIP*. Você pode baixá-lo junto o arquivo para download no endereço constante na seção *Links*.

Suas principais propriedades são:

- *AudioInDevice*, *AudioOutDevice* e *VideoInDevice*: indica os dispositivos de entrada de áudio, saída de áudio e entrada de vídeo, respectivamente. São configurados chamando o método *ShowMediaSelectionDialog*;
- *ConnectTimeout*: tempo de desconexão em segundos;
- *EnablePreview* e *EnableVideo*: habilitam e desabilitam o *preview* do vídeo que será transmitido e o vídeo da chamada;
- *PreviewWindow* e *VideoOutVideo*: apontam para componentes descendentes de *TWinControl* onde serão mostrados os vídeos, normalmente utilizamos um *Panel* para isso.

Seus principais eventos são:

- *OnConnect*: conexão estabelecida;
- *OnDisconnect*: conexão interrompida;
- *OnFail*: conexão falha;
- *OnIncomingCall*: recebe uma chamada;
- *OnStatus*: muda o status (conectado, desconectado, ligando etc).

Para instalar o componente basta descompactar o arquivo correspondente a sua versão do Delphi em uma pasta. Abra o arquivo *BlitzVoIP.dpk* no Delphi 7, clique no botão *Install* para instalar o componente. No Delphi 2005/2006 você deve clicar com o botão direito sobre o pacote *BlitzVoIP.bpl* no *Project Manager* e escolher a opção *Install*. Não esqueça de adicionar a pasta onde você descompactou o componente no *Library Path*.

Nota: O componente é *freeware*.

A aplicação

Feche tudo e crie uma nova aplicação. Veja na **Figura 4** uma sugestão para o formulário principal.

Altere o nome do formulário para "frmVidPhone". Altere a seção *private* do formulário de acordo com a **Listagem 1**, criando os métodos e implementado os mesmos.

O método *Add* adiciona uma mensagem no *ListBox* e na *StatusBar*. É para podermos acompanhar cada passo da comunicação. O método *RefreshDeviceLabels* atualiza os *Labels* com o nome dos dispositivos de áudio e vídeo selecionados bem como os *CheckBoxes*.

Adicione no evento *OnCreate* do formulário o seguinte código:

```
BVVoIP1.CallCallingPartyID := edtMyName.Text;
btnConnect.Enabled := True;
btnDisconnect.Enabled := False;
RefreshDeviceLabels;
Add('Pronto...');
```

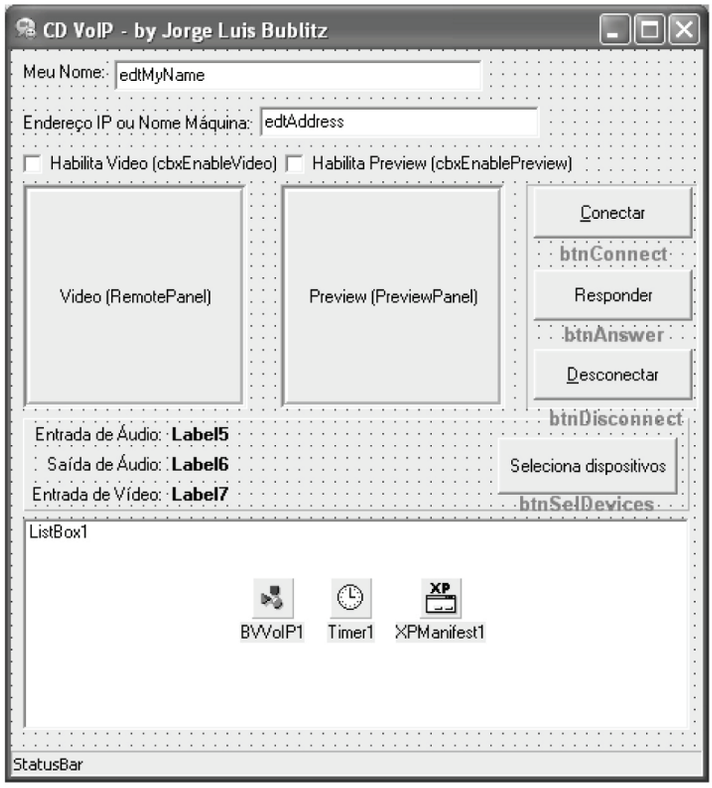


Figura 4. Nosso formulário principal

Listagem 1. Métodos do nosso formulário

```
...
private
  procedure Add(const S: string);
  procedure RefreshDeviceLabels;
...
procedure TfrmVidFone.Add(const S: string);
begin
  ListBox1.Items.Add(S);
  StatusBar1.SimpleText := S;
end;

procedure TfrmVidFone.RefreshDeviceLabels;
begin
  cbxEnablePreview.Checked := BVVoIP1.EnablePreview;
  cbxEnableVideo.Checked := BVVoIP1.EnableVideo;
  Label5.Caption := BVVoIP1.AudioInDevice;
  Label6.Caption := BVVoIP1.AudioOutDevice;
  Label7.Caption := BVVoIP1.VideoInDevice;
end;
```

CallCallingPartyID

A propriedade *CallCallingPartyID*, apesar do nome estranho, está de acordo com o padrão H.323. É através dela que você se identifica numa chamada VoIP.

Altere agora o evento *OnTimer* do *Timer1* e os eventos *OnClick* dos botões *Conectar*, *Desconectar*, *Responder* e *Seleciona dispositivos* de acordo com a **Listagem 2**.

O evento *OnTimer* é utilizado para tocar um som quando chamamos ou quando recebemos uma chamada. Para isso, usa-se a propriedade *Tag* do componente. Os arquivos *wave* estão no arquivo para download do artigo. O evento *Connect* do *BlitzVoIP*, como você deve ter percebido, tenta estabelecer uma comunicação VoIP com o endereço passado.

Listagem 2. Eventos OnTimer e OnClick dos botões

```
procedure TfrmVidFone.Timer1Timer(Sender: TObject);
begin
  { Adicione MMSystem em uses }
  if Timer1.Tag = 1 then
    PlaySound('ringin.wav', 0, SND_ASYNC)
  else
    PlaySound('ringout.wav', 0, SND_ASYNC);
end;

procedure TfrmVidFone.btnConnectClick(
  Sender: TObject);
begin
  Timer1.Tag := 0;
  Timer1.Enabled := True;
  btnConnect.Enabled := False;
  cbxEnablePreview.Enabled := False;
  cbxEnableVideo.Enabled := False;
  Add('Conectando...');
  btnDisconnect.Enabled := True;
  BVVoIP1.Connect(edtAddress.Text);
end;

procedure TfrmVidFone.btnAnswerClick(
  Sender: TObject);
begin
  BVVoIP1.Connect('');
  btnAnswer.Enabled := False;
  btnConnect.Enabled := False;
  btnDisconnect.Enabled := True;
  cbxEnablePreview.Enabled := False;
  cbxEnableVideo.Enabled := False;
end;

procedure TfrmVidFone.btnDisconnectClick(
  Sender: TObject);
begin
  Add('Desconectando...');
  BVVoIP1.CancelCall;
  cbxEnablePreview.Enabled := True;
  cbxEnableVideo.Enabled := True;
  btnAnswer.Enabled := True;
  btnConnect.Enabled := edtAddress.Text <> '';
  btnDisconnect.Enabled := False;
  Timer1.Enabled := False;
end;

procedure TfrmVidFone.btnSelDevicesClick(
  Sender: TObject);
begin
  if BVVoIP1.ShowMediaSelectionDialog then
    RefreshDeviceLabels;
end;
```

Se o argumento passado for vazio, o componente fica ativo, esperando uma chamada. E o evento *CancelCall* encerra a chamada. Para a nossa aplicação funcionar só precisamos digitar o código da **Listagem 3** no evento *OnIncomingCall* do *BVVoIP*.

Listagem 3. Evento OnIncomingCall

```
procedure TfrmVidFone.BVVoIP1IncomingCall(
  VoIP: TBVCustomVoIP;
  CallerAddr: string; var Accept: Boolean);
begin
  Timer1.Tag := 1;
  Timer1.Enabled := True;
  Add('Recebendo chamada de: ' + CallerAddr);
  Accept := (Application.MessageBox(PChar(
    'Você tem uma chamada vinda de ' +
    CallerAddr + '.'#13'Aceita?'), 'Mensagem',
    MB_YESNO+MB_ICONQUESTION) = IDYES);
  if Accept then
    Add('Respondendo...')
  else
    Add('Ok...');
  Timer1.Enabled := False;
end;
```

O parâmetro *CallerAddr* traz o nome da pessoa que está chamando, sendo a propriedade *CallCallingPartyID*. Já *Accept* é um parâmetro passado por referência, onde se o valor for *True* a chamada é aceita, senão recusada.

Executando

Ao executar o programa, basta você digitar o nome ou o IP da máquina que deseja chamar e clicar em *Conectar*. Na máquina chamada, será mostrada a mensagem da **Figura 5**.

Aceitando a chamada, é só falar, como se fosse um telefone convencional. Para encerrar, clique no botão *Encerrar*. Vale lembrar que se vocês tiverem *webcams* instaladas, o CD-VoIP se torna um verdadeiro "vídeo-fone".

> **Nota:** Obviamente, a máquina que está sendo chamada deve ter o projeto aberto e em execução, para aceitar a chamada.

Conclusão

Vimos neste artigo que a comunicação por VoIP não é um bicho de sete cabeças. O funcionamento da aplicação é bem simples e você fica a vontade para montar o seu próprio *Skype*. Uma boa sugestão é integrar o CD-VoIP com o Chat que está disponível no site da Clube Delphi. Até a próxima e boas compilações. ■

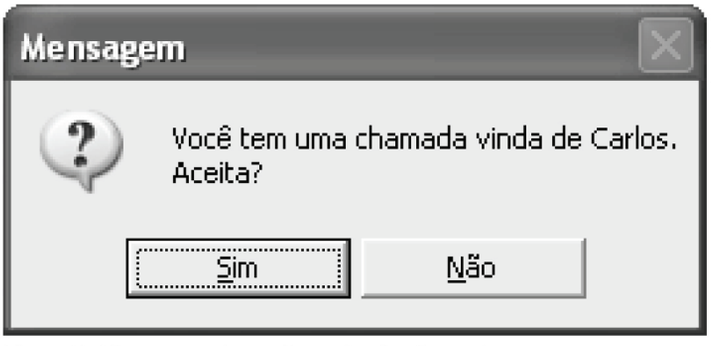


Figura 5. Mensagem de confirmação de chamada

Links

Minha página pessoal onde você pode baixar o Blitz VoIP.

bublitz.tripod.com/projetos.html

Informações técnicas sobre a TAPI.

msdn.microsoft.com/library/default.asp?url=/library/en-us/dntapi/html/msdn_tapi_30.asp

Automatizando backup no Firebird

RAFAELA DE CAMPOS
(rafaela@cmsi.com.br)
é desenvolvedora Delphi há 6 anos, graduanda em Administração pela UEM e trabalha como Gerente de Projetos na Strada Consultoria e Desenvolvimento, Software House que atua no segmento de transportes com aplicações n-camadas e bancos de dados PostgreSQL e Firebird.

Só quem teve um HD perdido com todas suas informações sabe o quanto dói a falta de um backup. Agora imagine perder dados de semanas ou meses de clientes, notas fiscais ou estoque?! Isso sim é um grande problema. Mas como ficar lembrando de fazer backup todos os dias? Somos seres humanos, sujeitos a esquecimentos e falhas, que podem acontecer nos piores momentos. Para evitar esse tipo de problema, existem ferramentas de automatização de backups específicas para cada tipo de banco de dados. Neste artigo abordaremos o FIBS (*Firebird-Interbase Backup Scheduler*), ferramenta *Open Source* (GNU/GPL) voltada para backup de bancos Firebird e InterBase.

> **Dica:** Confira nesta mesma edição um artigo que mostra como criar um utilitário personalizado para backup.

O FIBS

A primeira versão do FIBS foi lançada no início de 2005 e em julho de 2006 se tornou *Open Source* sob a licença GPL (*General Public License*). Até o fechamento deste artigo, está na versão 2.0.2. Possui várias características que o tornam atrativo, como facilidade de instalação e uso e arquitetura *multithread*. Algumas de suas principais características são:

- Suporta todas as versões do Firebird e InterBase. Isso acontece porque utiliza os próprios *Gbak* e o *GFix* do servidor, apenas disparando-os no horário desejado;
- Suporta até três *mirrors* em locais diferentes, incluindo endereços FTP;
- Suporta compressão de bases maiores que 4 GB;
- Suporta compressão no formato *GZip*;
- Pode enviar e-mail para um endereço pré-

configurando avisando erros no backup;
- É leve e pode rodar como aplicação ou serviço;
- Pode ser configurado para guardar os últimos backups por quantidade ou idade (data de criação);
- Possui interface intuitiva e tem uso simples;
- Tem seu código-fonte disponível para download;
- Pode ser usado com os clientes conectados ao banco.

Usando o FIBS

Depois de instalado, vamos iniciar o FIBS através do menu *Iniciar>Programas>FIBS Firebird-Interbase Backup Scheduler 2.0.2>FIBS 2.0.2*. Por padrão ele vem com dois exemplos de agendamento de backup, um acessando o banco localmente e outro remotamente (**Figura 1**).

Nosso primeiro passo é configurar o caminho do *GBak*, para que a ferramenta possa funcionar. Isso pode ser feito tanto pelo botão *Prefs* quanto pelo menu *View>Preferences* (**Figura 2**).

Em *GBak Directory* aponte para o diretório *bin* da instalação do Firebird ou InterBase, onde encontram-se o *GBak* e o *GFix*. Em *Log Directory* aponte para o diretório desejado onde ficarão os logs dos backups efetuados. Na opção *Backup Priority* definimos a prioridade do processamento do FIBS durante o processo de backup. E, quanto aos e-mails, para que o FIBS possa enviá-los, é necessário que os parâmetros do servidor SMTP estejam preenchidos.

Pronto! Feitas as devidas configurações vamos agendar um backup para testar seu funcionamento. Para isso na tela principal, iniciaremos uma nova tarefa no menu *Task>New Task*, e preenchê-la como mostrado nas **Figura 3**, **4** e **5**.

> **Importante:** Os diretórios apontados nas propriedades *Backup Dir*, *Mirror Dir 1*, *Mirror Dir 2* e *Mirror Dir 3* deverão ter sido previamente criados, pois nesse momento o FIBS não permite criá-los nem aceita diretórios inexistentes.

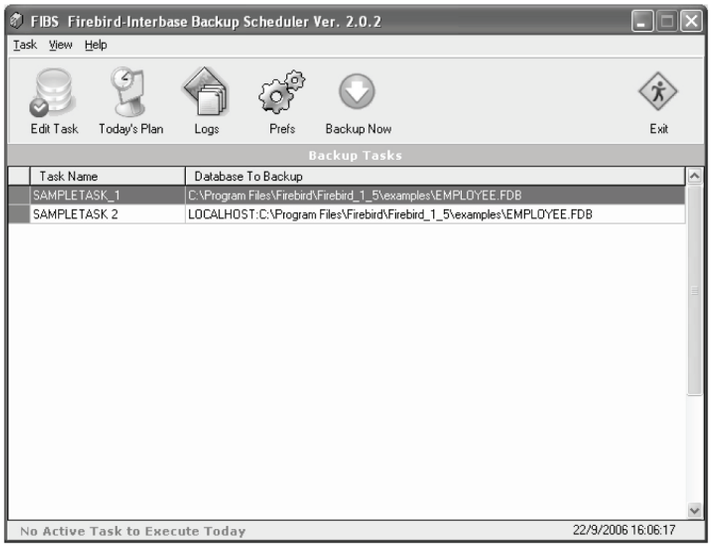


Figura 1. Tela inicial do FIBS

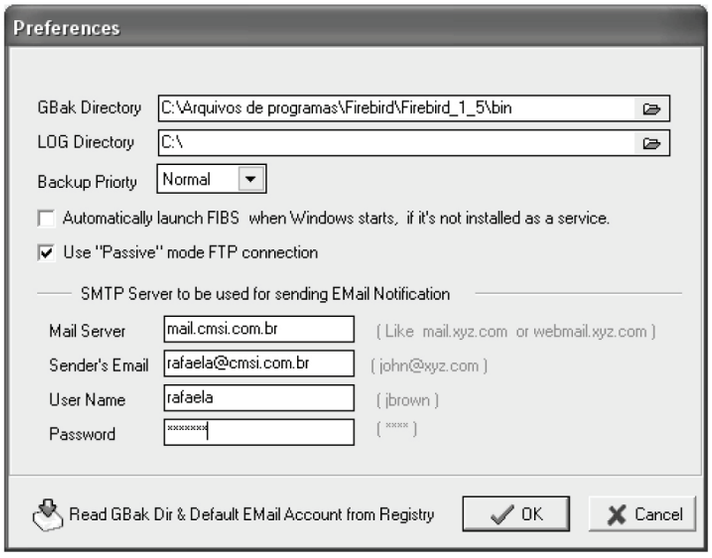


Figura 2. Tela de configurações do FIBS

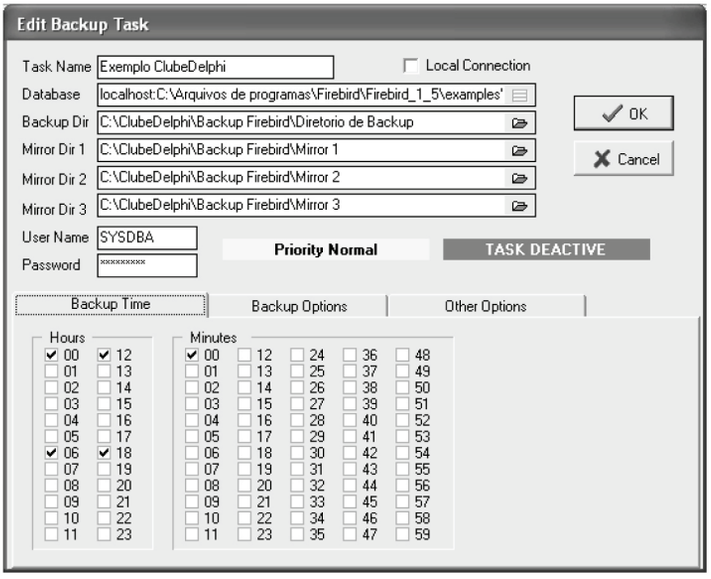


Figura 3. Nova tarefa de backup (aba Backup Time)

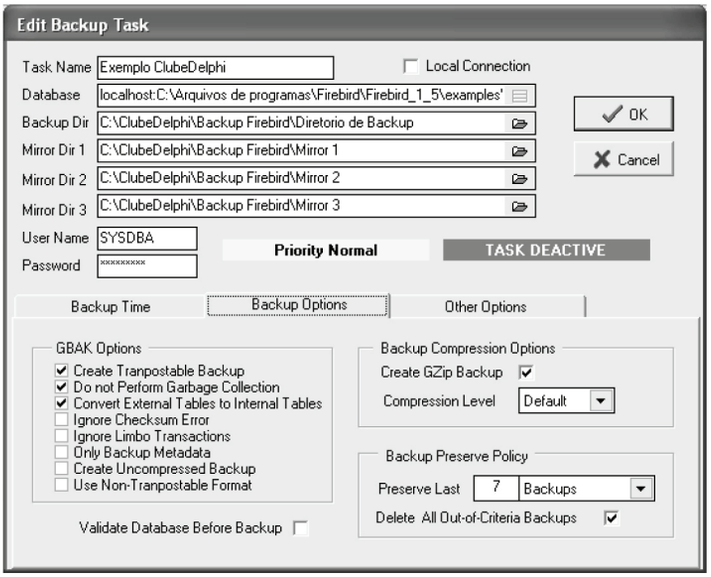


Figura 4. Aba Backup Options do editor de tarefas

Nota: Não marque a opção *Local Connection*, use "localhost" antes do caminho do banco, como mostra a **Figura 3**).

Na **Figura 3**, além de preencher o nome da tarefa, o caminho do banco de dados e os diretórios de backup e *mirrors* (opcionais) também definimos o usuário e a senha do banco, que por padrão são: "SYSDBA" e "masterkey".

Na aba *Backup Time* escolheremos quando serão efetuados os backups, a composição da hora é formada pelas horas e minutos marcados, no caso do exemplo, serão efetuados quatro backups por dia, sendo nos horários: 00:00h, 06:00h, 12:00h e 18:00h. Na aba *Backup Options* (**Figura 4**), podemos escolher as opções do *GBak* a serem executadas, como seguem:

- *Create Transportable Backup*: backup transportável entre plataformas e versões de servidor;
- *Do not Perform Garbage Collection*: não efetua a limpeza do lixo do banco, ou seja, fragmentos de dados perdidos;
- *Convert External Tables to Internal Tables*: transforma os arquivos externos acessados em tabelas do banco;
- *Ignore Checksum Error*: ignora erros de checagem de integridade dos dados;
- *Ignore Limbo Transactions*: ignora as transações que ficaram abertas quando o banco foi encerrado ou caiu;
- *Only Backup Metadata*: somente faz backup da estrutura do banco, ignorando seus dados;
- *Create Uncompressed Backup*: cria o backup descomprimido;
- *Use Non-Transportable Format*: somente para casos onde o backup será restaurado na mesma plataforma e mesma versão de servidor.

Além disso, podemos escolher se será feita validação do banco antes do backup (*Validate Database Before Backup*), se compactaremos o backup com o *GZip* e seu nível de compressão, e quantos backups ou qual o tempo máximo de backups que armazenaremos (*Backup Preserve Policy*).

Para finalizar a criação da tarefa, temos outras opções (*Other Options*, mostrada na **Figura 5**), onde pode ser configurado o endereço de e-mail que receberá os avisos de erro no backup, assim como especificar algum *script* ou programa que deverá ser executado após a rotina.

O último passo é ativar a tarefa, para isso, na listagem de tarefas na tela principal, escolha a tarefa e acesse o menu *Task>Activate "Exemplo ClubeDelphi"*. Agora, é só esperar o horário específico ou efetuar o backup manualmente através do botão *Backup Now*.

Outros recursos

Para visualizar as tarefas agendadas e ativas, acesse o menu *View>Backup Executing Times in Today* ou pelo botão *Today´s Plan*. Serão mostrados todos os horários e bancos a serem efetuados os backups no dia (**Figura 6**).

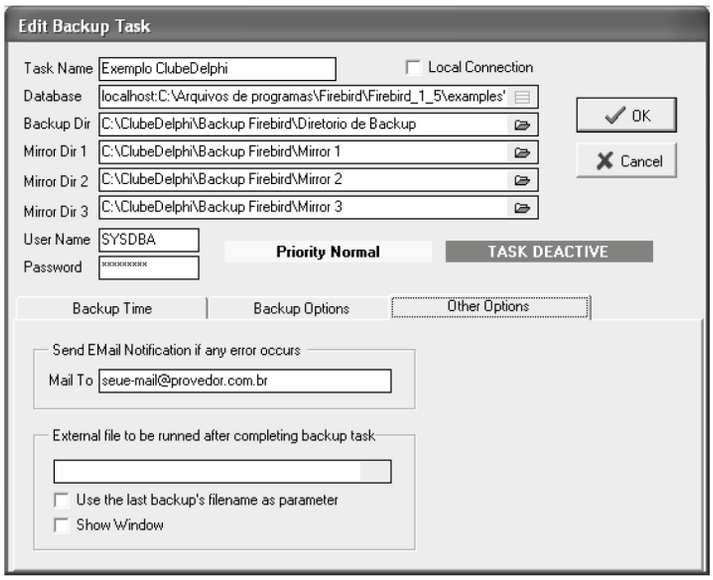


Figura 5. Indicando o e-mail na aba Other Options

Para visualizar somente os horários diários da tarefa específica, devemos acessar *View>Backup Time Settings of Task "Exemplo ClubeDelphi"* (**Figura 7**).

Observação: Para que uma tarefa possa ser alterada, necessita previamente estar desabiltada, isso é feito através do menu *Task>Deactivate "Nome da Tarefa"*.

Conclusão

Existem outros agendadores de backup para o Firebird/InterBase, como o *GBak Scheduler*, além do abordado neste artigo. Porém por características como: facilidade de uso e interface agradável e intuitiva, o FIBS foi escolhido, visto que o mecanismo do backup ocorre da mesma forma em ambos.

Ao final deste artigo é importante novamente ressaltar a importância dos backups e, além disso, de backups em lugares diferentes, pois de nada adianta ter 10 cópias de um mesmo arquivo se todas estiverem no mesmo lugar. ■

Links

Site do FIBS

www.talatdogan.com

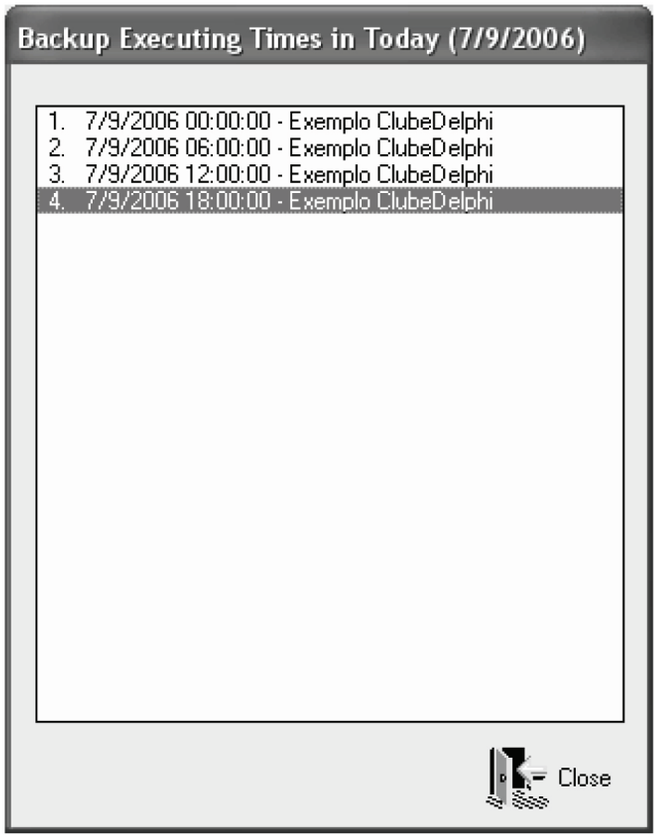


Figura 6. Planejamento de tarefas do dia

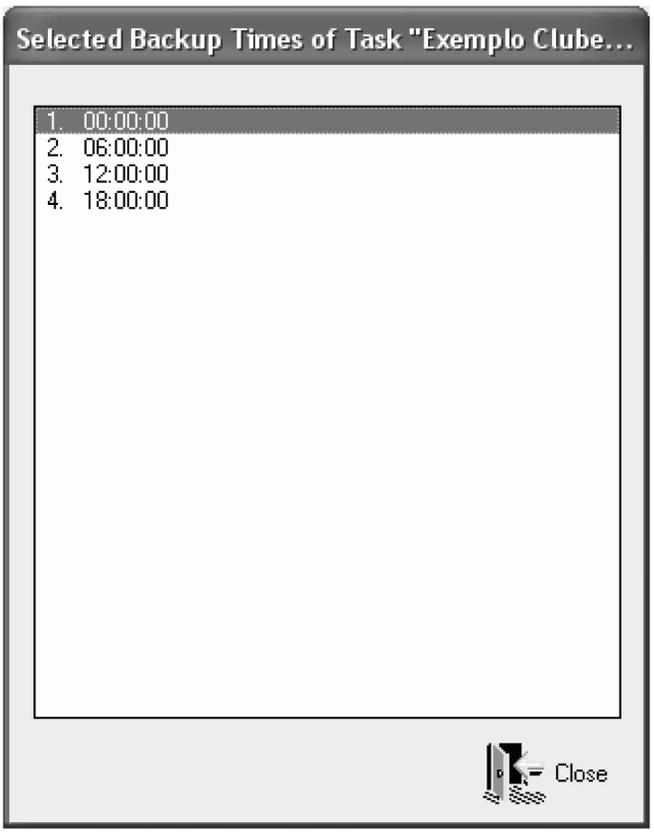
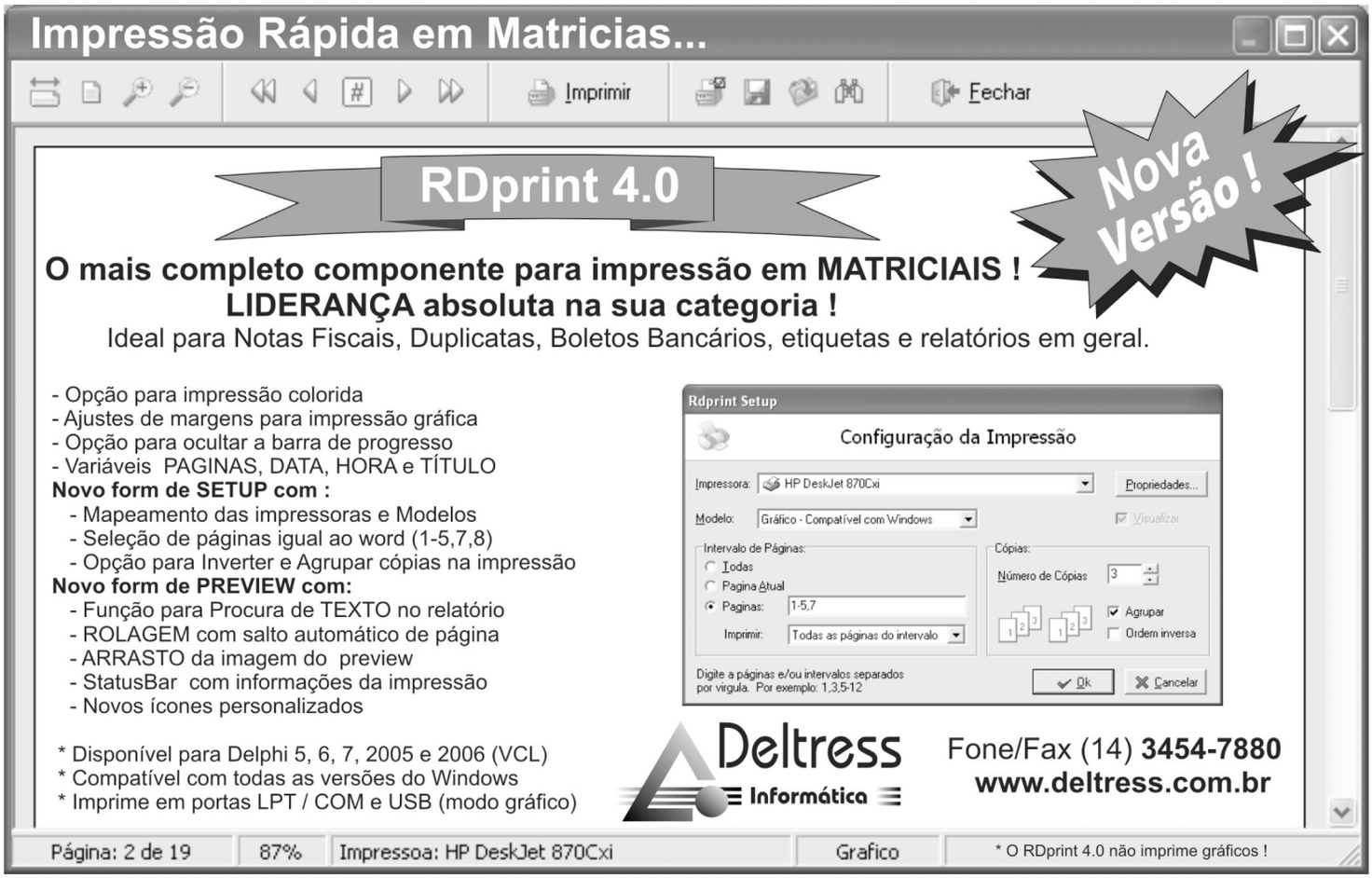


Figura 7. Horários diários da tarefa "Exemplo ClubeDelphi"



Serialização de objetos

DANILO ROCHA VALENTE
(valente@ntier.com.br)
é analista programador, e trabalha com a linguagem Delphi desde a versão 1.0 e é DBA Oracle.

FERNANDO NOMELLINI
(nomellini@ntier.com.br)
é analista programador, e trabalha com a linguagem Delphi desde o turbo Pascal 7.

Podemos dizer que serializar significa colocar os valores dos atributos de um objeto de uma forma que fique em série (seqüência). A serialização é uma técnica usada para transmitir objetos entre camadas, rotinas e até mesmo aplicações. Pode-se também utilizar essa técnica para persistir objetos, ou seja, gravar objetos em disco, fazer a transmissão remota de objetos via rede, armazenar os objetos em um banco de dados e/ou arquivos (XML, TXT etc.).

Se você faz a serialização, naturalmente vai querer fazer o processo inverso, que seria restaurar os atributos de um objeto gravado em um *stream*, que pode ser um arquivo XML, TXT etc. Chamamos esse processo de desserialização.

Cenário atual

Normalmente, quando precisamos armazenar ou transmitir objetos em Delphi, escrevemos um método específico para exportar e importar os atributos de cada classe que estaremos manipulando. Dessa maneira, além de utilizarmos uma abordagem para cada situação, o que se torna dispendioso e pouco produtivo, nos preocupamos com uma implementação que não faz parte da regra de seu negócio, assim como o fato de termos uma rotina de exportação e importação para cada classe, aumentando muito a chance de erros no seu projeto.

A ausência de um padrão para o formato de exportação/importação dificulta a integração de novos membros na equipe, que se veêm obrigados a entender o que o programador anterior fez no código para cada caso.

nTierObjects

nTierObjects é um framework para permitir a comunicação das diferentes camadas de uma aplicação (camada cliente, camada de aplicação ou de regra de negócio, camada de banco de dados, etc.) através da utilização de objetos puros. Dessa maneira, a arquitetura abstrai detalhes de implementação, como

o acesso a banco de dados e a codificação de mensagens (de aplicação) entre as camadas do sistema, permitindo que o desenvolvedor se concentre na implementação das regras de negócio de suas aplicações em uma estrutura 100% Orientada a Objetos.

Como primeiro passo, a empresa criadora do nTierObjects está disponibilizando um componente *free* e *open source* chamado *ntoStream*, para serialização automática de objetos, em formato XML, de forma a criar um padrão que possibilite a abstração da implementação do processo de serialização e desserialização.

Esse componente somente serializa objetos descendentes da classe *TPersistent*. Os atributos que se deseja serializar devem estar declarados na seção *published* da classe, devido a uma limitação dos recursos de reflexão da própria linguagem. Além disso, ele exporta os itens das classes descendentes de *TList*. Toda classe que se deseja serializar deve estar registrada. Veremos como fazer isso no exemplo mais adiante.

Vantagens de utilização do ntoStream

Como primeiro motivo, podemos citar a facilidade de uso e ganho de produtividade, pois o desenvolvedor não precisa mais se preocupar com a implementação de rotinas específicas para cada uma de suas classes. Além disso, haverá uma padronização no formato em que o objeto é serializado, o que facilita a compreensão por qualquer membro da equipe, antigos ou novos.

O *ntoStream* possibilita também a redução de erros, pois ele se encarregará de serializar todos os seus objetos da mesma maneira. Garante a interoperabilidade entre aplicações, ou seja, um objeto criado por uma aplicação pode ser utilizado em outra (imagine esse uso em aplicações n Camadas, onde se tem um servidor de aplicação e um *client* conversando puramente via objetos. Lembre-se que quando utilizamos um *ClientDataSet*, estamos trabalhando de modo relacional, e não Orientado a Objetos).

Baixando e instalando o componente

Acesse o link *www.ntier.com.br/wiki/index.php/NTo:Downloads*. Baixe o arquivo *NTierObjects.zip*, salve-o e descompacte-o em uma pasta qualquer (sugerimos o nome "nTierObjects"). Para instalar, abra o arquivo *NTierObjects.dpk* que se encontra na pasta *dpk/D7* ou *dpk/D2006*, de acordo com a versão do Delphi utilizada. Compile e instale o pacote. Adicione no *path* do Delphi o caminho para a pasta *\dpk\D2006\dcu* ou *\dpk\D7\dcu* e *src*.

Exemplo de utilização

Vamos criar um aplicativo para demonstrar a capacidade de serialização e desserialização do *ntoStream*. Lembre-se, o componente foi desenvolvido inicialmente para trabalhar com o Win32. Assim, crie um novo projeto no Delphi e adicione no formulário, um *ntoStream* que se encontra na aba *NtierObjects*. Crie uma nova unit e digite o código da **Listagem 1**.

Salve a unit como "PessoaImpl.pas". Note que as classes descendentes de *TPersistent* devem ser registradas na seção *Initialization*, utilizando o *RegisterClass(Classe)* e que as classes descendentes de *TList* devem ser registradas utilizando o *Clas-*

sManager.RegisterServerClass(classe) (*ClassManagerImpl* deve estar na cláusula *uses* de sua unit).

A unit declara uma classe do tipo *TPessoa*, uma classe *TTelefone*, uma classe de lista de *TTelefone* (*TTelefones*) e um tipo enumerado *TTipoTelefone*. Note que uma pessoa contém uma lista de telefones, que podem ser de um determinado tipo (*ttCelular* ou *ttFixo*).

Quando o *ntoStream* serializar o objeto *TPessoa*, automaticamente ele exportará os atributos da pessoa e também a lista de telefones, convertendo inclusive os tipos enumerados para o XML final. Vamos agora utilizar a unit *PessoaImpl* e o *ntoStream* no formulário. No formulário, coloque dois *Memos*, dois *Edits* e dois botões, de modo a ficar de acordo com a **Figura 1**.

No evento *OnClick* do botão *Exportar*, digite o código da **Listagem 2**.

No evento *OnClick* do botão *Importar*, digite o código da **Listagem 3**.

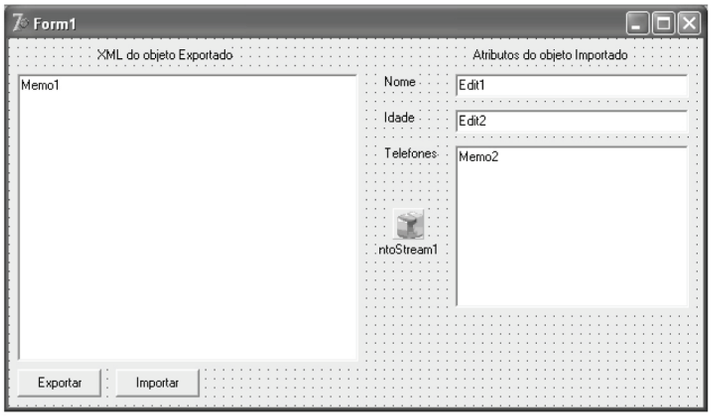


Figura 1. Componentes adicionados no formulário da aplicação

Listagem 1. Descrição das classes Pessoa, Telefone e Telefones

```
unit PessoaImpl;
interface
uses
  SysUtils, Classes, ntoStreamImpl, ClassManagerImpl;
type
  TTipoTelefone = (ttCelular, ttFixo);
  TTelefone = class(TPersistent)
  private
    FPessoaOID: string;
    Fnumero: string;
    Ftipo: TTipoTelefone;
    procedure Setnumero(const Value: string);
    procedure SetPessoaOID(const Value: string);
    procedure Settipo(const Value: TTipoTelefone);
  public
  published
    property PessoaOID: String read FPessoaOID
      write SetPessoaOID;
    property numero: string read Fnumero
      write Setnumero;
    property tipo: TTipoTelefone read Ftipo
      write Settipo;
  end;

  TTelefones = class(TList)
  private
    function GetItem(const Index: Integer): TTelefone;
  public
    destructor Destroy; override;
    property Items[const Index: Integer]: TTelefone
      read GetItem; default;
  end;

  TPessoa = class(TPersistent)
  private
    FIdade: Integer;
    FNome: string;
    FOID: string;
    FTelefones: TTelefones;
    procedure SetIdade(const Value: Integer);
    procedure SetNome(const Value: string);
    procedure SetOID(const Value: String);
    procedure SetTelefones(const Value: TTelefones);
  public
    constructor Create; reintroduce;
    destructor Destroy; override;
  published
    property OID: string read FOID write SetOID;
    property Nome: string read FNome write SetNome;
    property Idade: Integer read FIdade
      write SetIdade;
    property Telefones: TTelefones read FTelefones
      write SetTelefones;
  end;

implementation
{ TPessoa }
constructor TPessoa.Create;
begin
  inherited Create;
  FTelefones := TTelefones.Create;
end;
destructor TPessoa.Destroy;
begin
  FTelefones.Free;
  inherited Destroy;
end;
procedure TPessoa.SetIdade(const Value: Integer);
begin
  FIdade := Value;
end;
procedure TPessoa.SetNome(const Value: string);
begin
  FNome := Value;
end;
procedure TPessoa.SetOID(const Value: string);
begin
  FOID := Value;
end;
procedure TPessoa.SetTelefones(
  const Value: TTelefones);
begin
  FTelefones := Value;
end;

{ TTelefone }
procedure TTelefone.Setnumero(const Value: string);
begin
  Fnumero := Value;
end;
procedure TTelefone.SetPessoaOID(const Value: string);
begin
  FPessoaOID := Value;
end;
procedure TTelefone.Settipo(
  const Value: TTipoTelefone);
begin
  Ftipo := Value;
end;

{ TTelefones }
destructor TTelefones.Destroy;
var
  i: Integer;
begin
 for i := 0 to Count - 1 do
    Items[i].Free;
  inherited;
end;

function TTelefones.GetItem(
  const Index: Integer): TTelefone;
begin
  Result := Inherited  Items[Index];
end;

Initialization
  RegisterClass(TPessoa);
  RegisterClass(TTelefone);
  ClassManager.RegisterServerClass(TTelefones);
end.
```

Listagem 2. Código do evento OnClick do botão Exportar

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Pessoa: TPessoa;
  auxTelefone: TTelefone;
begin
  Pessoa := TPessoa.Create;
  try
    { Dados da pessoa }
    Pessoa.OID := '001';
    Pessoa.Nome := 'Severino da Silva';
    Pessoa.Idade := 29;
    { Objeto telefone 1 }
    auxTelefone := TTelefone.Create;
    auxTelefone.PessoaOID := '001';
    auxTelefone.numero := '(11)5555-5555';
    auxTelefone.tipo := ttFixo;
    Pessoa.Telefones.Add(auxTelefone);
    { Objeto telefone 2 }
    auxTelefone := TTelefone.Create;
    auxTelefone.PessoaOID := '001';
    auxTelefone.numero := '(11)9999-9999';
    auxTelefone.tipo := ttCelular;
    Pessoa.Telefones.Add(auxTelefone);
    { Aqui pedimos ao ntoStream que serialize o
      objeto Pessoa e o resultado é escrito no Memo1 }
    Memo1.Lines.Text := ntoStream1.ToXML(Pessoa);
  finally
    Pessoa.Free;
  end;
end;
```

Listagem 3. Código do evento OnClick do botão Importar

```
procedure TForm1.ImpoClick(Sender: TObject);
var
  Pessoa: TPessoa;
  i: Byte;
  Tel: ShortString;
begin
  Memo2.Clear;
  { Criando uma nova pessoa }
  Pessoa := TPessoa.Create;
  try
    { Pedimos para o ntoStream popular os atributos
     da pessoa, baseado no XML passado. }
    ntoStream1.FromXML(Pessoa, Memo1.Lines.Text);
    { Exibimos os atributos do OBJETO pessoa que
     acabamos de criar e que foi populado. }
    Edit1.Text := Pessoa.Nome;
    Edit2.Text := IntToStr(Pessoa.Idade);
    { Sabemos quantos telefones a pessoa tem }
    if Pessoa.Telefones.Count > 0 then
     { E para cada um deles }
     for i := 0 to Pessoa.Telefones.Count - 1 do
     begin
       { Verificamos o tipo (ENUMERADO) }
       case Pessoa.Telefones[i].tipo of
         ttCelular: Tel := 'Celular: ';
         ttFixo: Tel := 'Fixo :';
       end;
       { E exibimos seu número }
       Tel := Tel + Pessoa.Telefones[i].numero;
       Memo2.Lines.Add(Tel) ;
     end;
  finally
    Pessoa.Free;
  end;
end;
```

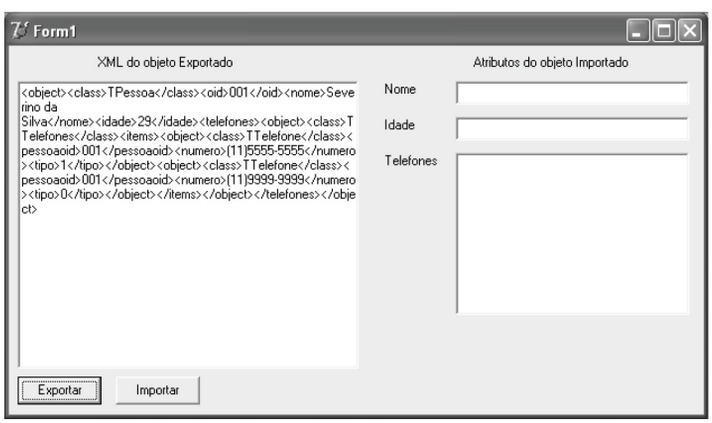


Figura 2. Exportando o objeto para XML

Links

Site da nTier Objects

www.nTier.com.br

Execute o projeto e clique em *Exportar*. Deve aparecer algo parecido com a **Figura 2**.

Note que você pode até alterar o texto, por exemplo, trocando o atributo *Nome* ou *Idade*, mas nunca altere a estrutura do arquivo e as *tags*. Clique em *Importar* e terá algo semelhante ao mostrado na **Figura 3**.

Opções do ntoStream

Podemos agora falar das propriedades *ExecucaoEmCascata*, *ShowExportErrors* e *ShowImportErrors*, conforme segue:

- *ExecucaoEmCascata*: do tipo *Boolean*. Com o valor *True* por *default*. Indica se o componente deve exportar/importar todos os atributos que são objetos. No exemplo ele exportou a pessoa e sua lista de telefones, com *ExecucaoEmCascata = False*, seriam exportados somente os atributos *Nome* e *Idade*;
- *ShowExportErrors* e *ShowImportErrors*: do tipo *Boolean*. Com o valor *False* por *default*. Quando ativos, indicam quais classes não foram possíveis exportar/importar, indicando o motivo. Normalmente é causado por uma classe que não foi registrada.

Conclusão

Pudemos observar no exemplo que é muito simples implementar a serialização de objetos utilizando a ferramenta apresentada. Com o amadurecimento do desenvolvimento profissional Orientado a Objetos, essa certamente será uma técnica amplamente utilizada, como já existe em linguagens em que o desenvolvimento deve obrigatoriamente ser Orientado a Objetos, como Java, onde podemos encontrar ferramentas similares.

Além disso, esse componente aumenta a produtividade daqueles que fazem uso desse tipo de implementação manualmente. Gerando o resultado da serialização em XML com uma estrutura padronizada, o *ntoStream* permite ampla conectividade entre ferramentas, linguagens e sistemas. ■

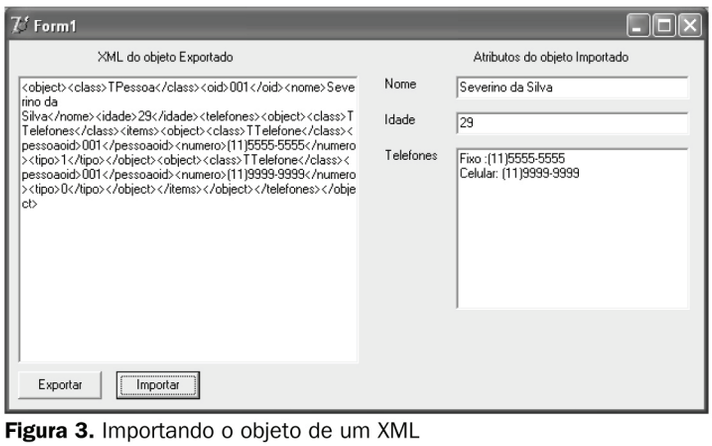


Figura 3. Importando o objeto de um XML

Criando uma ferramenta personalizada para backup no IB/FB

RODRIGO OTTO MOSTAERT
(otto3d@gmail.com)
é Analista/Programador Delphi. Trabalha com desenvolvimento Win32 e .NET com banco de dados MySQL e Firebird em projetos cliente/servidor e multicamadas. Especializado em Sistema WMS. Possui cursos oficiais Borland pela Facilit (www.facilit.com.br).

Veremos neste artigo, como criar um projeto que permite a criação de backup manual ou agendado de um banco de dados IB/FB. O projeto é bastante simples, utilizaremos componentes nativos do próprio Delphi. Nele solicitamos as informações básicas para que possa ser feita a cópia de segurança, que são: banco de origem, o nome e caminho do arquivo de destino (arquivo FBK), usuário e senha do banco, e por fim a hora em que será feito o backup.

Para configurar o sistema de backup, tanto automático e/ou manual, precisamos informar alguns campos (citados anteriormente). Após, não precisaremos mais informar os dados, quando o sistema for fechado os dados serão gravados no arquivo de configuração que será criado junto ao executável. Ao desenvolver o exemplo, utilizei como banco de dados o Firebird 1.5 versão *SuperServer*.

Desenvolvimento

Na **Figura 1** temos a tela do projeto, podendo ser modificado de acordo com as necessidades que o desenvolvedor tiver.

A codificação é bastante simples, pois usaremos a classe *TIBBackupService* da unit *IBServices* para gerar a cópia do banco de dados, conforme a **Listagem 1**, no método *FazBackup*.

No *OnCreate* do formulário (**Listagem 2**) criaremos um *Timer* que fará a agendamento (a hora) do backup que será definido no sistema. No evento, ao criarmos o *Timer*, informamos que a cada segundo, o mesmo executará o método *TempoVerificador*, que verifica se a hora do computador é a mesma do campo *Hora* do agendamento. Assim, quando a hora configurada chegar, o backup do banco será gerado.

Como melhoria, você pode colocar o projeto na barra de tarefas do Windows, onde o sistema verificará a hora do backup. Como foi dito anteriormente, os dados informados nos campos do sistema, serão salvos dentro do arquivo de configuração pra que não haja o retrabalho toda vez em que o sistema for aberto.

Listagem 1. Código do método FazBackup

```
with TIBBackupService.Create(Self) do
begin
  ServerName := 'Servidor';
  { Nome do Servidor }
  Protocol := Local;
  LoginPrompt := False;
  Params.Clear;
  Params.Add('user_name='+ editUsuario.Text);
  Params.Add('password='+ editSenha.Text);
  try
    Active := True;
  except
    on E: Exception do
    begin
      { Caso ocorra um erro, é exibida a mensagem de
        erro e abortada a execução }
      Mensagem('# Ocorreu um erro ao tentar conectar',
        true);
      Mensagem('Mensagem de Erro: '+ e.Message, True);
      tTempoVerificaDor.Enabled := True;
      Screen.Cursor := CrDefault;
      Free;
      Abort;
    end;
  end;
  try
    Verbose := True;
    { Opção de Restore }
    Options := [NonTransportable];
    { Opções de extração dos dados,
      podendo trazer apenas o metadados }
    DataBaseName := editBancoOrigem.Text;
    { Nome do banco a ser feito o backup }
    BackupFile.Clear;
    { Limpa a lista dos arquivos do backup }
    BackupFile.Add(editBancoDestino.Text +'_'+
      FormatDateTime('dd-MM-yyyy HH_mm', now)+'.fbk');
    { Arquivo que será o backup }
    ServiceStart;
    { Iniciar o Backup }
    while not EoF do
    begin
      Mensagem(GetNextLine,
        CheckBoxMostrarDetalhe.Checked);
      { Pega linha por linha do processo para mostrar
        em um Memo }
      Application.ProcessMessages;
    end;
  finally
    Active := False;
  end;
  Free;
end;
```

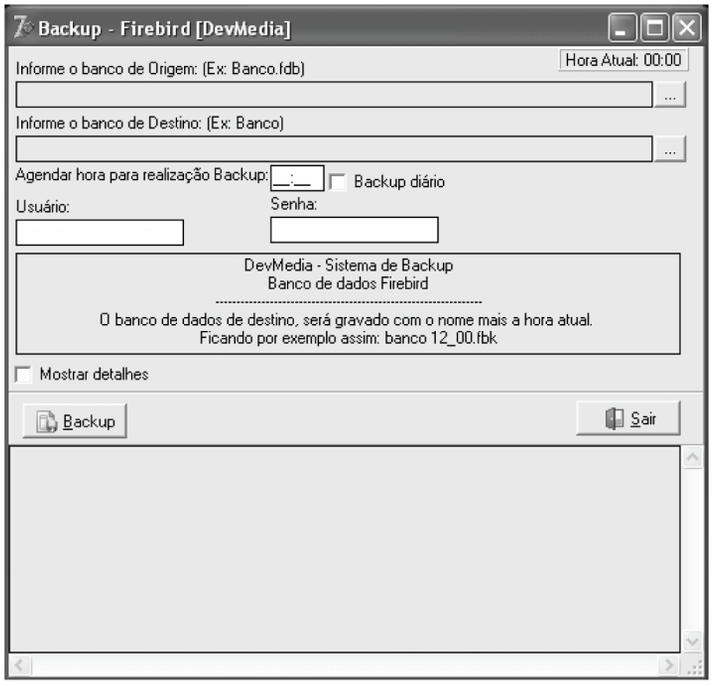


Figura 1. Tela do projeto de backup

“ O Maior Portal para Desenvolvedores da América Latina.”

Acesse:

www.devmedia.com.br

DevMedia GROUP

Listagem 2. Timer de agendamento do backup

```
{ No private do formulário, declare a
  variável Timer e o evento de verificação
  Declare no uses ExtCtrls }
  private
    tTempoVerificaDor: TTimer;
  procedure TempoVerificador(Sender: TObject);

{ No OnCreate teremos a criação do objeto e o
  preenchimento dos controles }
procedure TFrmPrincipal.FormCreate(Sender: TObject);
begin
  { Carrega os dados do arquivo config.ini }
  editBancoOrigem.Text := TFuncoes.LerArquivoINI(
    ExtractFilePath(Application.ExeName)+'config.ini',
    'BANCO', 'Origem');
  editBancoDestino.Text := TFuncoes.LerArquivoINI(
    ExtractFilePath(Application.ExeName)+'config.ini',
    'BANCO', 'Destino');
  editHora.Text := TFuncoes.LerArquivoINI(
    ExtractFilePath(Application.ExeName)+'config.ini',
    'BANCO', 'Hora');
  editUsuario.Text := TFuncoes.LerArquivoINI(
    ExtractFilePath(Application.ExeName)+'config.ini',
    'BANCO', 'Usuario');
  editSenha.Text := TFuncoes.LerArquivoINI(
    ExtractFilePath(Application.ExeName)+'config.ini',
    'BANCO', 'Senha');
  { Cria o Timer que verificará a hora do Backup }
  tTempoVerificaDor := TTimer.Create(Self);
  tTempoVerificaDor.Interval := 1000;
  tTempoVerificaDor.OnTimer := TempoVerificador;
  tTempoVerificaDor.Enabled := True;
end;

{ Método TempoVerificador }
procedure TFrmPrincipal.TempoVerificador(Sender: TObject);
begin
  { Atualiza a hora do Panel e faz a verificação da Hora programada }
  PanelHora.Caption := 'Hora Atual: '+
    FormatDateTime('HH:mm', now);
  if TFuncoes.ValidaHora(editHora.Text) then
  begin
    if (FormatDateTime('HH:mm', now) = editHora.Text) and
      (CheckBoxBackupDiario.Checked) then
      FazBackup;
  end;
end;

{ Função de leitura dos dados do arquivo de
  configuração que recebe como parâmetro o arquivo INI
  que será lido, o tópico e o sub-tópico }
class function TFuncoes.LerArquivoINI(AArquivo,
  ATopico, ASubTopico: string): string;
var
  ArqINI: TINIFile;
begin
  { Declare em uses IniFiles }
  if not FileExists(AArquivo) then
    result := ''
  else
  begin
    ArqINI := TIniFile.Create(AArquivo);
    result := ArqINI.ReadString(ATopico, ASubTopico, '');
    FreeAndNil(ArqINI);
  end;
end:
```

No evento *OnClose*, usaremos um procedimento de gravação dos dados no arquivo (**Listagem 3**).

Listagem 3. Gravando as informações no arquivo INI

```
procedure TFrmPrincipal.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  { Salva os dados no arquivo INI }
  TFuncoes.EscreverArquivoINI(ExtractFilePath(
    Application.ExeName)+'config.ini', 'BANCO',
    'Origem', editBancoOrigem.Text);
  TFuncoes.EscreverArquivoINI(ExtractFilePath(
    Application.ExeName)+'config.ini', 'BANCO',
    'Destino', editBancoDestino.Text);
  TFuncoes.EscreverArquivoINI(ExtractFilePath(
    Application.ExeName)+'config.ini', 'BANCO',
    'Hora', editHora.Text);
  TFuncoes.EscreverArquivoINI(ExtractFilePath(
    Application.ExeName)+'config.ini', 'BANCO',
    'Usuario', editUsuario.Text);
  TFuncoes.EscreverArquivoINI(ExtractFilePath(
    Application.ExeName)+'config.ini', 'BANCO',
    'Senha', editSenha.Text);
  { Liberamos os objetos da memória }
  FreeAndNil(tTempoVerificaDor);
  Action := caFree;
  FrmPrincipal := nil;
end;
```

O método *EscreverArquivoINI*, recebe como parâmetro o arquivo de configuração que será salvo os dados, o tópico, subtópico e o valor a ser gravado (**Listagem 4**).

Listagem 4. Método para salvar o arquivo INI

```
class procedure TFuncoes.EscreverArquivoINI(
  AArquivo, ATopico, ASubTopico, AValor: string);
var
  ArqINI: TINIFile;
begin
  ArqINI := TIniFile.Create(AArquivo);
  ArqINI.WriteString(ATopico, ASubTopico, AValor);
  FreeAndNil(ArqINI);
end;
```

Todos os métodos de validação, escrita e leitura do arquivo INI, entre outros usados no projeto, encontram-se no arquivo *UFuncoes.pas* do projeto.

Conclusão

Vimos como é fácil e prático desenvolver um sistema bastante útil atualmente, não só para os desenvolvedores, quanto também para os responsáveis pelos sistemas de uma empresa, afinal, todos precisamos de cópia de segurança hoje em dia. Até a próxima! ■

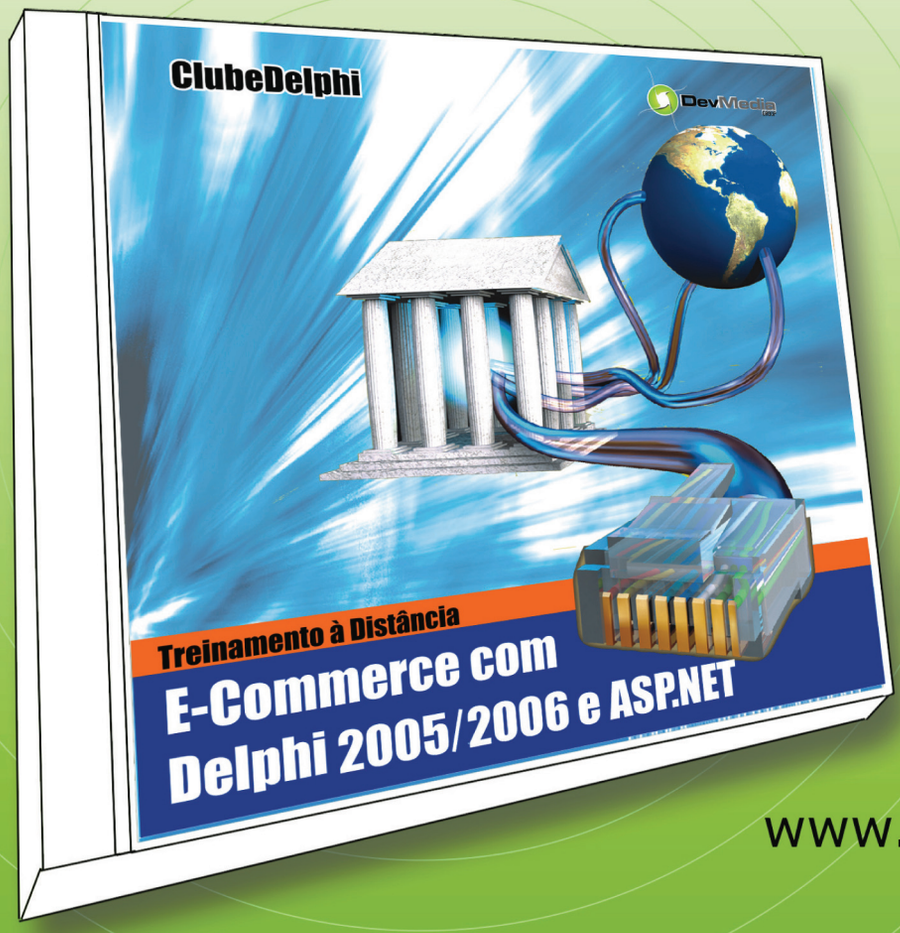
CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como trabalhar com o *ZipMaster*, componente para compactação de arquivos, muito usado em sistemas de backup.

www.devmedia.com.br/articles/viewcomp.asp?comp=1293

Treinamentos a distância.

Aprenda de forma prática, eficiente e econômica.



E-Commerce com Delphi 2005/2006 e ASP.NET

Aprenda a desenvolver poderosas aplicações Web através do Delphi 2005/2006 e as tecnologias ASP.NET, WebBroker, Intraweb e WebSnap.
O curso conta com módulos textuais;
35 horas de vídeo-aulas;
fórum exclusivo para os alunos e suporte on-line.

COMPRE JÁ O SEU!

www.devmedia.com.br/curso

Outros Cursos

Delphi e Relatórios

Com este curso você desenvolverá relatórios através do Delphi 7, do Delphi 2005 e Delphi 2006, utilizando as ferramentas Rave Reports, Quick Reports e Crystal Report. Além do módulo textual o curso contém mais de 20 horas de vídeo-aulas; fórum exclusivo e suporte

Evento à distância - Introdução ao .NET

Conheça os recursos do ASP.NET com o Visual Studio 2003 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a IDE do VS; componentização; DataGrid e criar aplicações de BD com o ADO.NET! São 6h de palestras em formato de vídeo-aulas e você pode tirar suas dúvidas com os palestrantes!

Evento à distância - Novidades do Delphi 2005

Conheça os recursos do Delphi 2005 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a nova IDE do Delphi; utilizar programação OO no Delphi 2005; trabalhar em equipe com o novo Starteam e criar aplicações de BD com o ADO.NET!

DevMedia

Quantas cópias de seu software existem no mercado?

cópia

original

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com